Ryszard Tadeusiewicz
Tomasz Gaciarz, Barbara Borowik & Bartosz Leper

Discovery

of Neural Network Properties

by means of C# programs

Acknowledgments

Authors address many thanks to
M.Sc. Jakub Tutaj
for many years valuable cooperation

Content

10. Self-organizing neural networks

    10.1. What is the structure of the neural network, at which you will create mappings, being result of self-organizing?

    10.2. What is the self-organization in the network and what it might be helpful for?

    10.3. How to implement a neighborhood in a network?

    10.4. What follows from the fact that some neurons we consider neighbor?

    10.5. What can Kohonen networks do?

    10.6. What will Kohonen Networks do in case of a more difficult data?

    10.7. What happens in a network with excessively wide range of initial weights?

    10.8. Can I change the form of self-organization in the course of a network self-learning?

    10.9. Alright, but what it all might be useful for?

    10.10. How the network can serve as a tool for transformation of an input space dimension?

    10.11. Control questions and self-study tasks

11. Recurrent networks

    11.1. What is recurrent neural network?

    11.2. What features have network with feedback?

    11.3. Who needs this kind of networks ‘with loops’ ?

    11.4. How Hopfield`s network is constructed?

    11.5. How does the neural network work as an associative memory?

    11.6. How works program for investigating Hopfield network by yourself discovery?

    11.7. A few interesting examples

    11.8. How and why we can use automatic patterns generation Hopfield network?

    11.10. What more it is worth observe in associative memory?

    11.11. Control questions and self-study tasks

Preface for English version

(Written by Ryszard Tadeusiewicz, rtad@agh.edu.pl)

The book which I am introducing to you constitutes the third attempt to work out the topic indicated in the title. I want to believe that it is the most successful one.

The first attempt was a series of articles, popularizing neural networks themes on pages of the popular computer monthly magazine ***Enter***. This magazine was printed in Poland but it was read willingly also in a few neighboring countries, for example in Slovakia. My series of articles were quite well accepted by their Readers. It was going this way up to the end of 1994 when the editors of this magazine had turned to me with the request. They asked me to try to write an article explaining the Readers of this magazine in a maximally popular way what these neural networks are about that they were becoming so famous. As a reply instead of one article I wrote the entire cycle of them which was appearing in all next numbers of the ***Enter*** magazine from January 1995th till March 1996th inclusive. They printed these articles in a few internet portals (willingly I gave my assent to this) and they was read until now, what is attested by numerous letters coming from Readers.

Seeing the success of these articles the editor-in-chief of the Academic Publishing House, Prof. Leonard Bolc, suggested to collect these articles and publish them in a book form. Preparing and publishing this book lasted for a little while because material for the clenched tome had to be more integrated and tidied up than it was necessary while writing from one month to the next consecutive fragments of the cycle of these articles. At the end a book was released:

Tadeusiewicz R.: Elementary introduction into neural networks with demonstration programs, Academic Publishing House, Warsaw, 1998

It is possible to show that book as the immediate predecessor of the current book, handed over at that moment to the hands of Polish Readers. The book „***Elementary introduction*** …" has very quickly disappeared from bookshops shelves, bought out in a sequence during only one month. Few copies of it, available now at many libraries, have been worn out and destroyed while overused through many years by very numerous readers. The ones which still stay at libraries are practically on the border of physical annihilation. Numerous requests from Readers have become the direct cause of making an attempt to prepare the next release of this book. However it has turned out that progress in the field of neural networks has been so fast that after a few years from the first publication the „***Elementary introduction*** …" book already has become so much outdated. Therefore instead of the next edition (corrected and supplemented) of that book I have decided to present Readers the brand new book - the very one which you are holding in your hand.

What justifies such a decision?

To explain this more precisely let us examine the motivations which accompanied preparing the first articles for the monthly magazine ***Enter***, and the motivations which are accompanying publishing the new book on the same topic here and now.

The articles (and then the book being the result of their compilation) have come into existence at the request of right editors (the editors of the monthly magazine ***Enter*** and of the Academic Publishing

House). These requests were settled in the peculiar context of those years. It is worthwhile here to remind in short this context because it has a connection both with the form and with the substantive contents of this book. So exactly in the beginning of the 90's an event took place which even in a dynamically changing world of the computer science happens very rarely. A fashion came (if not to say – epidemic!) of applying to everything so-called „Neurocomputers". This fashion lasts as a matter of fact until today and it is probably justifying the fact that you are reading this book. Admittedly a little from the 90's have changed: after a certain time (when the first tide of uncritical enthusiasm has already died down), the futurist name Neurocomputer had been left and the appropriate computer tools have started to be called more sensibly and more calmly, simply as neural nets, but the fashion for applying them has lasted still and what's more it has intensified, what is possible to trace in the P.1 picture.

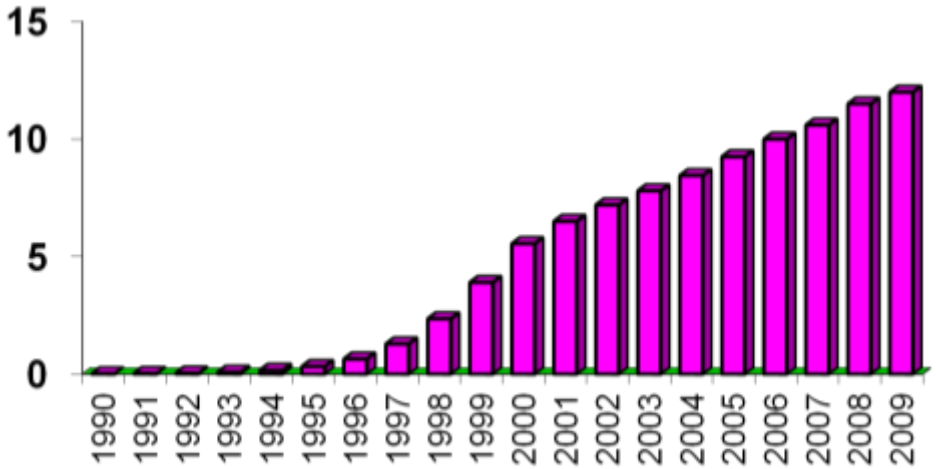


Fig. P.1 Profits (in the millions USD) coming from providing the sale software for creating and using neural networks. Given for the American market.

The reason of the described event was simple: specialists of different fields in the 90's „discovered" that neural networks could constitute an unusually effective tool for computer solving their problems. I am writing „discovered" in inverted commas, because these „miraculous" properties of the new tool had been already well known earlier. But at first they had been familiar to not very numerous specialists: to biocyberneticians, and saying more precisely – to neurocyberneticians. Then in the nineties the same properties of neural networks had been enthusiastically described and discussed in many specialized computer writings.

In Poland at that time I was a chief specialist in neural networks as for at least 20 years I had been dealing with computer modeling of human brain. Additionally willingly and often I have popularized the knowledge about this subject. Moreover, at the beginning of the nineties I wrote the first in Poland book about neural networks which until today has been read and quoted. At present this book is possible to study all over the world since it has been made available in the sets of the Polish Internet Library at the following address: http://www.pbi.edu.pl . It is also available in the sets of the Academic Digital Library at the http://abc.agh.edu.pl address. Because this book is often read, there is also a direct immediate access to its content, possible through its own Internet address: http://winntbg.bg.agh.edu.pl/skrypty/0001 .

However the recalled book has been meant for professionals. Whereas in the 90's there were many people who were simply interested in neural nets, but they did not have professional preparation in this field. There was not any appropriate literature for them. It was exactly me that the recalled earlier editors had asked me to write the first Polish popular book on neural networks theme.

I am not concealing that the fact that it was I who had been asked for explaining to the wide whole of a public what neural networks are – gave me a big satisfaction. Therefore I undertook this work. I treated it as a task for one evening at first, because seemingly what could be more simple than popular explaining what one's knew well and very much liked. Meanwhile it turned out that issues about neural networks that I wanted to tell about – are many more than I could expect. As a result popularizing neural networks has become for me a fascinating but also very time-consuming occupation for the next consecutive years. When starting the work alone I didn't realize how many fascinating things about these networks could be said (and shown in practice!) to interested in those things Readers. But then the topic was expanding under my pen and was occupying the next issues of the monthly magazine. Essentially my articles on this topic had been appearing incessantly for more than one year! Then, the same material, but gathered and tidied up, became the base for writing the previous version of the book.

At first I was going to explain to Readers mainly this that in properties of neural networks which fascinate most people there have been mirrored only some (and very minor) fragments of this vast and extremely interesting neurological knowledge. This knowledge had been accumulated for many decades by researchers who were trying to observe and keep up with information transformation and learning processes within human and animals brains (fig. P.2). Through this knowledge they were getting into the core information about forming intelligent behaviors within hidden secluded parts of the brain.

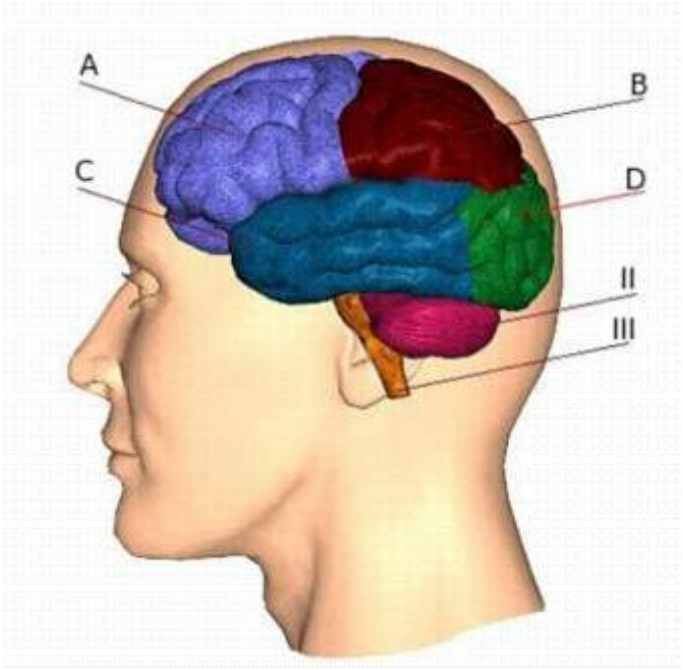


Fig. P.2. Neural networks constitute the computer imitation of some detected properties while examining the brain of people and animals

Works of these researchers resulted in the entire series of scientific discoveries. Many researchers of the brain have been honored with the most valuable trophy the scholar can get: with the Nobel Prize. In the P.1 table I am showing a list of those Nobel Prize winners who contributed to the understanding of principles, according to which the biological mastermind is functioning, and by this they have created premises for building technical neural nets.

Table P.1: the Nobel Prizes connected with these examinations of the nervous system, of which results were exploited indirectly or directly in neural networks.

1904	Pavlov I.P.	theory of conditioned reflexes
1906	Golgi C.,	examining the structure of the nervous system
1906	Ramón y Cajal S.	discovering that the brain consists of the network of separate neurons
1920	Krogh S.	describing regulating functions in the body
1932	Sherrington Ch. S.	examining nervous steering of the work of the muscles
1936	Distances H., Hallett L.O.	discovering chemical transmission of nervous impulses
1944	Erlanger J., Gasser H. S.	processes in the single nerve fiber
1949	W.R Hess.	discovering the post of the midbrain
1963	Eccles J.C., A.L Hodgkin., A.F Huxley.	mechanism of the electric activity of the neuron
1969	Granit R., Hartline H.K., Wald G.	physiology of the vision
1970	Katz B., Von Euler U., Axelrod J.	transmission of the humoral information in nerve cells
1974	Claude A., De Duve Ch., Palade G.	examining of the structural and functional organization the cell.
1977	Guillemin R., Schally A., Yalow R.	examining of hormones the mastermind
1981	Sperry R.	discoveries concerning the functional specialization of hemispheres of the cerebellum
1981	Hubel D.H., Wiesel T.	discovering principles of the processing of information in the visual system
1991	Neher E., Sakmann B.	functions of electrovalent channels in nerve cells

As it can be seen from the table, on the beginning of the 90's biologists already knew quite a lot about the brain. Biocyberneticians for years have been building cybernetic models (mathematical, simulational and computational ones) of all these mechanisms, which had been detected in the brain. By the way it has been found beyond all doubt that the mastermind collects and processes an

information more efficiently and more wisely, than computer systems. The brain is better than computers, though the latter ones are surpassing it in terms of an action's speed and a capacity of mass memory.

After ascertaining that a cybernetic model of the brain is associating an information better than a computer program, is classifying more efficiently, recognizes and searches compound sets of data (for example images) more intelligently on the basis of their content and associations occurring between them etc. – it was obvious that basing on similar mechanisms the functioning of useful computer tools had been only a matter of time. In the 90's the novelty was mainly this that engineers, researchers and practitioners who had not known them before had learned about these possibilities. Scientists dealing with the neurophysiology and cognitive science have been writing about them for years. Therefore sensational tone of many news and reports on (significant!) successes obtainable thanks to neural networks was accepted by many specialists of biocybernetics with a certain dose of a forgiving understanding. However, since this phenomenon took a really mass scale and moreover it was notable and was waking a wide interest (among others of readers of the recalled monthly magazine ***Enter***) therefore appeared a request for writing something about it - and I did it.

But let us leave this historical motif on a side and let us proceed to discuss the genesis and the content of this book which at the moment you are holding in the hand.

From the perspective of the 21st century first decade's knowledge it can be stated that the fashion for applying neural networks which started growing in the 90's, lasts until today. It doesn't result only from the universally known and constantly observed tendency of computer specialists for being fascinated by every next news, brought by progress of technology and the development of civilization. The admiration for neural networks also had and still has its additional, concrete and important causes. The main reason for these networks huge popularity have been really excellent results, obtainable with the help of these new tools while solving many problems, about which for ages it had been obvious that they were particularly difficult. Sensational news reports of pioneers, who as first tried neural nets out and described their outstanding properties in their books and publications, have encouraged numerous followers. Throughout the whole decade of the 90's and during the first years of the XXI century very often were appearing reports about this who and for what purpose with a success had used these nets. Therefore while writing at that time one of my next articles on this topic I had used the humorous expression that at that time the ignorance of neural networks in some spheres had started to be treated as a kind of *faux pas.*

So that you don't treat these assurances as baseless generalities, now I will try to show you, where and to what neural networks are being applied. This will allow to get the overall view on why they have such interesting properties. Look at the P.3 picture. In this picture I depicted (in a quite conventional way) a classification of tasks which could be performed by different computer systems. It is probably obvious for you that between them are simpler and more difficult tasks. And so on the horizontal axis of the shown graph I have marked some measure of the level of difficulty of a problem. Frankly speaking the accurate calibrating of this axis wouldn't be simple because it is not entirely obvious, in what way one should measure this difficulty. But certainly it is possible to distinguish simple tasks (more close to the left side of the graph) and difficult tasks (on the right-hand side). At the moment it is enough for us.

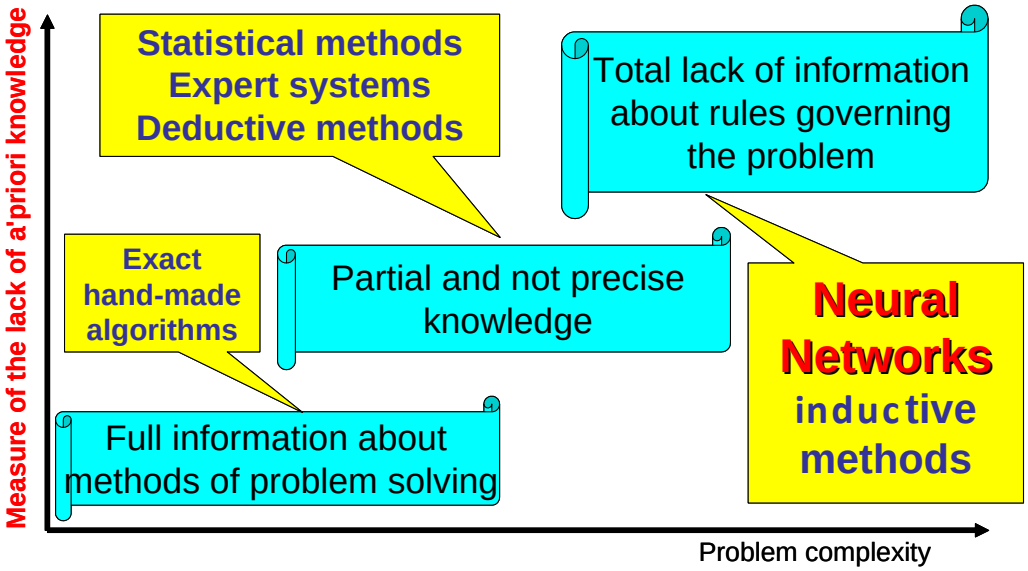


Fig. P.3. The characterization of computer tasks of varying difficulty and the location of these tasks, for the realization which peculiarly well are sending oneself neural networks.

A level of difficulty of a problem to be solved is not the only measure of troubles with which a computer specialist will have to deal. A second dimension of a problem that builds up its solution is the availability of a wisdom on which it is possible to rely. For some tasks their rules are exactly determined, although the number of these rules or the degree of their complication could cause that it would be still difficult to solve them. For example the author of a software which he writes for a large bank must work quite hard although the rules there are being known well and they are simple enough. When a customer is giving a money to a bank, then the state of his account should be increased and when he is taking money from the bank, then it is necessary to decrease the balance. But there is a huge number of such customers, and this causes that the program controlling all transactions for this bank is complicated and expensive. A physicist figuring out mathematical models describing the interior part of the atom has a different type of difficulty. The number of equations here is small but they are very hard to solve. In both quoted examples however the measure of complications, what here is an acquaintance of rules - is equal to zero, as the rules are known and they could be used to construct the tools solving these two problems.

We know how to use computers while solving tasks characterized by a full knowledge of rules. On the basis of these rules it should be built the algorithm and on the basis of this algorithm one has to write the adequate program. I don't claim that writing such a program is always easy and simple but at least it is exactly obvious what one should do as well as how one should do it.

The situation however could be more complicated. We must often solve problems, in which rules aren't well-known. If we want to solve them, we can sometimes do this based on the observation that processes which they concern are in some way repeatable. In such case we are dealing with the situation symbolically located in the central part of the P.3 picture. The acquaintance of rules isn't in this case complete and rules that are known aren't to the end certain – but in spite of these gaps we can use the comfortable technique of the deduction: We try to produce a general rule (or more often a set of general rules) which then we use to solve every concrete detailed problem.

Programs built for this class of problems usually take into consideration the fact that our knowledge is uncertain and incomplete, therefore in such cases we are willingly using statistical tools (for example in economics or agrotechnology) or rely on non-algorithmic knowledge. For example while solving such objectives we often rely on a common-sense wisdom of experts. Such a knowledge, based on years of experience, can be surprisingly effective. For example it is being used in popular expert systems suggesting diagnoses or advising the most appropriate therapy methods in medicine.

However there may exist an even more complex situation, in which one can't give any rules. In such problems the only thing that we have in our disposal is a number of examples of tasks which have been solved correctly. These correct answers can come from observations of the behavior of a system, whose properties we want to model.. We are building a model of the system, although we don't know what its internal structure is and why it is acting, but we can see – how. Textbook answers which we will want to imitate are sometimes given by somebody who can do it. Such an expert alone often doesn't know how he is achieving the required result but he does, what is necessary, and it works well.

At first perhaps you the Reader think that there are no such situations or that there are few of them. You are not right! Our mastermind is solving such tasks non-stop, for example in a connection with tasks defined in the psychology as perceptions. Subjectively the matter is very simple: here you are walking along a street and you are meeting a friend. You recognize her easily. You can see her face and so you do not have any doubts, who it is. But imagine that a computer carries out the same activity. For example to let some people into your house, and another not. The beginning is encouragingly simple, because at present there are universally available digital and video cameras. They are producing an image in the form which can be easily entered into a computer (see the P.4 fig.). However a question appears, what is next?

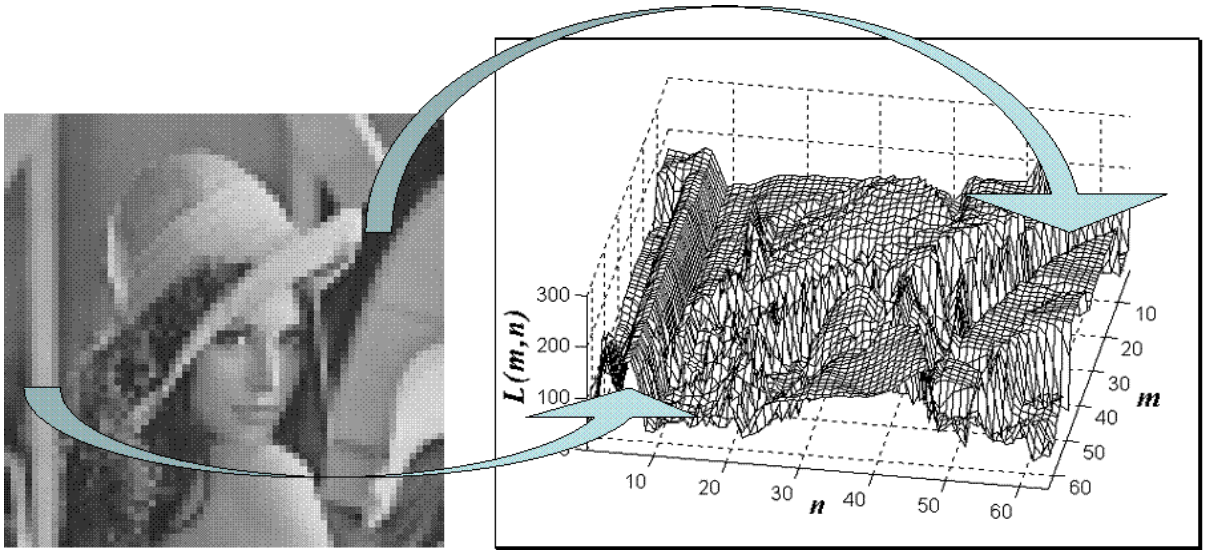


Fig. P.4. The digital image loaded into the memory of the computer can be considered as the two-dimensional function and easily it is possible to find characteristic elements of the image in the form of details of this function. It is hard however to give the algorithm of recognizing the person to the function presented in a picture on the basis of this function values only.

An image inserted into the computer memory is broken to millions of points (pixels). Brightness or color of each of these points are encrypted in a digital form. This causes that the digital image can be easily worked on the computer, as after all it is a calculating machine. What's more, it is possible to easily become convinced that in these millions of numerical values, corresponding to individual pixels there is included an information needed to recognize the introduced in the picture person. Because it is sufficient to show on a screen or print on a printer in order for someone to state beyond all doubt that: „*This is Lena*".

Okay, but how to give the algorithm, which these millions of values describing the grayness of individual points of the image, will convert to the same claim, but automatically generated by a computer? What's more, this algorithm must be able to recognize a familiar face, independently of whether it has been photographed "en face", whether from a profile, whether is more or less close, whether is lighter or darker etc. It is necessary to have a method, so that changeable elements of facial expressions (for example the presence or the lack of a smile), changeable headgear, a different background etc. - would not disturb us in effective recognizing a given person, (see – P.5). For our mastermind these matters constitute no problem but when we want to program this our ability in a form of a general algorithm for a computer, it appears that we aren't able to do this!

Fig. P.5. The same face can differently look in a picture depending on the illumination (a), depending on the location (b) and depending on the facial expression (c). It is making it difficult very much to set automatic recognizing!

In similar problems, while handing over a certain ability to other people we demonstrate many examples. For example we are showing a child pictures and we are saying: *This is a letter a, that letter is also A, though is smaller. And here you also have the letter A, which looks a little bit differently because is printed with different font - but it is being read the same.* Acting this way we count on this (and in general rightly) that the taught recipient of our message from these many

detailed demonstrations is able to produce for himself some synthetic way of thinking. This taught way of thinking will let it effectively recognize all shown model forms of considered images. What's more the child will be also able to recognize different models of the same letter. It concerns also recognizing a familiar person or some other object. While teaching people or training animals we are obtaining very good results in their abilities to generalize the knowledge, which they have received from the presentations of concrete examples. The taught child or the trained dog are also able to solve problems which earlier have not been shown them directly – with the condition that they would be enough similar to examples demonstrated to them during the learning process.

The required in this case attempt to solve a problem is called the inductive method. It uses the fact that unlike the earlier discussed deduction, in this case we **do not know any general rule**, on the basis of which we could solve concrete detailed problems. Instead of it, however, we have a handful of examples of concrete tasks, for which the correct answers are known. The mind of a human being is endowed with the talent which allows to analyze such concrete examples, to draw conclusions from them, and then this way to generalize these conclusions. The brain of the trained animal, for example a rat in a maze, also is able to learn rules of finding the appropriate way. In the end the rat is managing to solve not only the problems which have been earlier demonstrated to it as examples, but also different ones. We are saying that after the learning process the rat is more clever. A human being also. What about a computer?

As I recalled, specialists dealing with neurocybernetics have known already earlier the possibilities of providing the knowledge by teaching. This huge potential has always lied in the acquainted by biologists abilities of the brain (people's and animals') to learn and in neural methods of information processing. However the majority of remaining scientists and practitioners, especially the ones who for years have been using a computer, thought that the acquaintance of the precise functioning of an algorithm is the necessary condition of automatic solving every problem. When it wasn't, they were seeking at least a strong enough deductive rule. They thought that it was the essential condition of getting an effective solution for every theoretical or applicative problem. When it has turned out that neural networks are able to solve different problems without programming, using only a method of inferential reasoning by analogy, this fact has been greeted with a great interest. When then additionally it had been found out that they by themselves could discover the unknown earlier rules of proceedings and could solve tasks, for which nobody would be able to give the way of solving them – a great enthusiasm took control.

Academic news reports were soaring up as fireworks and conclusions formulated in them have often pointed out on uncritical fascination of their authors with this new tool. This wave of effective and striking successes of neural networks have again aroused a hope that a new „philosopher's stone" had been discovered, sought without success by alchemists and astrologers by many centuries. Of course it is necessary to treat this last sentence as the metaphor, because they have never been expecting neural networks to be the tool that would exchange lead into gold, but equally naively they thought that with the help of this new technique it would be possible to solve everything and always.

Neurocyberneticians have already known earlier the abilities to learn and to generalize knowledge that are characteristic for systems modeling the structure and functioning of the human brain , but they have also been conscious that the possibilities which are enclosed in this new technique are limited. Therefore they have been referring to these academic reports avidly because human

ingeniousness in applying neural networks has turned out to be incredibly big, but also with a peace, because it has been obvious that after a tide of enthusiasm must also come a time of disappointment. However the „neophytes in neurocomputing" and outside commentators of their achievements, i.e. professional historians of science and journalists writing comments about a scientific development, have not known the moderation and while describing successive achievements they have been expressing astonishment connected with admiration. This admiration and surprise last until today, so I wrote this book in order to help you to share this admiration with others. You will be able to admire neural nets performing experiments on your home computer and you will not need to be … surprised when I show you exactly, how and why that all is acting.

If you are still hesitating whether to read this book, then I will tell you something more. I don't know in what field of knowledge you are studying or in which you would like to study, but I guarantee you that easily you will find it interesting to apply neural networks in this your field. I can solemnly assure you about it, because for many such observers of behaviors of neural networks absolutely amazing is how this technique has turned out to be very much **universal**. Equally enthusiastically admire the possibilities and advantages of neural networks engineers applying them for example for steering of robots, as well as bank officials, who are using them for chasing embezzlers. Neural networks have found their applications in astrophysics at modeling the beginning of the Universe and within a food industry at cooking cookies branch to check when they are already well done. In a short time it has been found out that it would be possible to use neural nets almost for everything and in almost every field they turn out to be more effective than traditional computer methods that have been applied in these fields for years. For many traditional researchers it has been shocking.

In the history of the development of neural networks there are a lot of registered surprises. If a neural network was able to aid in estimations of new farming methods of plants more effectively than practiced for years statistical calculations, then it was a surprise to agrotechnicians. If a neural network was able to control a chemical reaction more precisely than a computer model based on a balance of energy and mass, then it was amazing for technologists. If the forecast given by a neural network turned out to be more accurate than obtained using other methods of predicting future action prices or currency rates, then it was incomprehensible to experts using traditional methods of econometrics. If a neurocomputer guaranteed a better control over the automated process, than classical signal processors and the newest digital drivers, then this fact was raising an anxiety of control science engineers. It would be possible to endlessly list these specialists, for whom neural nets have opened brand new possibilities, but they also became the source of brand new challenges, because their current knowledge suddenly has turned out to be outdated. Don't let the progress in the field of neural networks to surprise you, but get to know it most quickly and most precisely – and this book will help you very much in it!

The task I have taken while writing this book has not been simple. Admittedly, when I was already setting about writing articles and the first version of the book I did have behind me close to 20 years of the research work at creating and constant improving of successive neural nets. But books written

by specialists, which I wrote earlier on this topic (essentially above 50 books) weren't appropriate to explain the phenomenon of neural networks in a simple and transparent way for not a specialist. My numerous scientific papers on this topic, which I published, also weren't fitting for it, because all without an exception they were addressed to specialists-biocyberneticians. All these publications assumed that a reader had a considerable knowledge about problems discussed in the articles. Whereas now on the same topic I was supposed to write the popular, i.e. legible and intelligible to everyone work in dust. It has seemed almost unfeasible.

Because how I was supposed to expect from a doctor fearing mathematics that he would get to know strongly mathematical and objectively rather difficult (even for computer specialists) the theory of neural networks, before he starts using them while diagnosing illnesses or while making plans for the therapy for his patients? Or could I persuade a bank manager, who would like to use the neural adviser while giving credits, to penetrate different subtleties of neuroanatomical and neurophysiological nature, constituting the notional and conceptual base for forming networks and for using them (especially teaching)? So while writing this book I have made an attempt to explain what neural networks are without using any mathematics, and in this entire book there is not a single mathematical formula. I had to also explain to persons who about biology have heard a little and a very long time ago, in what scope and in what sense neural networks could be treated as models of the real neural structures (particularly fragments of the human brain). As well I had to do this without carrying out the too complex and too extensive lecture from the scope of anatomy and physiology of the nervous system. I could not in addition go deep into details of the knowledge about the technique of neural networks, because they are (as all details of every field of knowledge) difficult and complicated, but at the same time I wanted that after studying my articles you would **really** know how neural nets work and why they are so very useful. So I decided that I would give you the possibility of independent **discovering** the properties of neural networks. You will do this with the help of more and more universally available computers, which certainly you are using - if not at home, then at least at school or at work. So I have written series of **programs**, constituting the integral part of this book, which will let you by yourself examine and **discover**, what neural nets are and how they work.

The effort of writing these programs was considerable, but necessary. Perhaps I have surprised you with this statement, because you know well about it, how many cool programs on different topics are in the Internet, so instead of writing the own ones – maybe it would be better to look for them in the Web.

Of course I know and I value web resources about neural networks that have been gathered and made available there. Therefore in one of the articles of the created in the ***Enter*** magazine cycle I gave a long list of familiar to me computer programs, enabling to structure and simulate simple neural nets on the PC class computers. I gave also a list of sources from where these programs could be obtained. Unfortunately, it was not a good solution. These programs have usually required more knowledge from users to possess than I have been able to provide to my Readers through written popular articles. Moreover, they generally were made available according to such a rule that for getting their full functionality it was necessary to buy their license (secondary school children and different hobbyists could not afford this), whereas free versions of these programs were very limited (e.g. demo types). Essentially applying these programs, which were available, was far from this what

I had wanted to provide to my Readers – namely a joy of independent experimenting with neural networks built by oneself.

In this situation it was necessary to write programs with which the Reader could play by himself, obtaining besides the theoretical knowledge also a practical experience connected with neural network issues. I made this and I attached these programs to the previous book (titled *Basic leading into neural networks with demonstration programs*) in a form of the floppy disk. At present these programs are universally available from the Web page

http://www.agh.edu.pl/dydaktyka/sieci_neuronowe/basic

and many persons are still using them.

Unfortunately programs in that version were written in the BASIC language. More precisely speaking – in its dialect used by the QBASIC interpreter. However this interpreter had passed away together with the MS-DOS operating system, and that was already quite a long time ago.

Looking from a time perspective it is possible to admit that it was a good solution. Above all, programs written in „old, but good BASIC" have it to themselves, that – like in a case of every interpreted language – a program is simply a text file, which you can examine (and modify!) in any word processor. Unfortunately, there is no rose without spikes. For running such a program the recalled interpreter is required - the QBASIC program. Yes, yes – if so far you have been not conscious of the fact that saying the truth a computer cannot simply run a text file, even if it contains the text of the program written in whichever sophisticated programming language – then you have been moved out of the error. Nowadays MS Windows or Linux systems don't have the in-built QBASIC interpreter (what at one time was a norm), so you must install it by yourself. It is also obtainable together with its textbook (as files named QBASIC.EXE and QBASIC.HLP) from the mentioned earlier Web page:

http://www.agh.edu.pl/dydaktyka/sieci_neuronowe/basic/

However QBASIC is unfortunately a primitive programming language and its possibilities are firmly limited. Today, when offering by programs colorful, user-friendly Windows interface has become a norm, using both the QBASIC environment and the programs written in, could be irritating.

The programs that are intended to make reading **this book** easier and more pleasant are written in the completely different technology. They are written in the C# language, which is a *compiled* language. It means that you will get ready programs which you can install on your computer and run by clicking at them – the same as you do with the majority of other programs. Since they have been written (by co-authors of this book, Tomasz Gaciarz, Barbara Borowik and Bartosz Leper) specially with the thought about the Windows system, they have a user-friendly and transparent interface. We will play the ones you can download for yourself (legally and for free!) from my Web page:

http://www.agh.edu.pl/tad

The more accurate information on this topic, as how to download these programs and when to use them, will be given in chapter 4.

There appears however a rationalized question: Why have we made up our mind to use this new technology? After all in the BASIC it was possible to see the text of programs „without lens" and to understand by oneself what they are doing and how, whereas the compiled programs written originally in C# aren't too adequate for „reading"!

Above all because the new programs are more simple in use. What's more, using them is adapted to habits and likings of modern computer users, accustomed to using programs with the help and other conveniences delivered to them by the Windows system. You like comfortable and fast pull-down menu selections , different inventive indicator lights, various colorful icons sensitive for clicking on by a mouse, etc. – and not diligent inputting data from a keyboard, what was the only form of a communication with programs in BASIC!

This that these modern and improved programs are compiled (so they are illegible), should not constitute any bigger problem. Anyway, the majority of Readers will simply want to run and experiment with these demonstration programs and will not want not to examine how they are built. Of these few fans, who want to look inside those programs,", will be perforce less. However even they in this book won't be omitted! On the mentioned before Web page

http://www.agh.edu.pl/tad

there also is an unabridged source code of all the programs written by us as well as a kit of tools allowing to look inside these codes and to analyze them. For braver and more experienced there are also tools allowing to modify the source codes of our programs, to compile and to run them on one's own. For this purpose can be used provided by the Microsoft company the free programming environment called the Visual Studio Express 2005. We are encouraging to install this free environment on one's own computer also those who only want to view the source code of our programs. Thanks to this using the programs that demonstrate selected problems which are described in the book - will be far simpler!

So as you see, the book and the resources placed on the indicated to you earlier Web page, you can use in three ways:

- If you are interested in neural networks only from the theoretical side and you don't feel like playing with any programs – then it will be enough that you read the content of the book.

- If you like to check by yourself, how this and that works, then you can download ready programs which will allow you to build neural networks and examine them on your computer. In this way you will link the theory (read in the book) with practice (resulting from having fun with our programs) and by this you will gain the twofold ability connected with the theory of neural networks and with methods of applying them.

- But if additionally you are an amateur and a fan of programming - then you will be able to examine exactly, how these our programs have been built and you will be able to correct them, to change and to improve them as much as you wish. We have nothing here to conceal!

Thanks to the fact that you take on yourself the difficulty of reading the text of our programs (and maybe also of working them through), then you will be able to understand how they function, and you will also be able to change something here and there. If you are willing to modify the source

code of our programs and you will try to check „what if …", then you will enter the elite circle of neural networks authors, which is much more honorable than the group of only users alone of these networks.

So I am inviting everyone desiring to do this to willingly go deep into next chapters of this book and through them (as well as by using associated with the book programs) to become acquainted with the fascinating world of neural networks. You will get to know these networks mainly as artificial intelligence tools, which have had and still have a lot of computer science applications. But you will also get to know them as interesting models of fragments of your own mastermind. Perhaps neural networks will help you to understand the secretive world of your own intelligence and a complicated psyche of different people.

1. An introduction to natural and artificial neural networks

1.1. Why is it worth to learn about neural networks?

(translation by Paweł Olaszek; pawelolaszek@tlen.pl)

This book is for you, dear reader, to learn in a nice and easy way what neural networks are, how and why they work and how and where to use them. If you hold this book in your hands, I believe that its what you want to find out. This book is thick so it needs a lot effort to **read it**. You can ask yourself a question: Is it worth it? If yes, then why ? Maybe it is better to put it away and play some computer games?

The simplest answer is: **Yes, It is worth** to study neural networks because they currently are in interest of many researchers and practitioners. With their help, people made many interesting discoveries and for sure they will lead to further achievements.

If those reasons still did not convince you need to spend some time studying neural networks then you can recognize them worth knowing, because can you hear a lot about them and we are observing a trend for them?

However are those reasons enough to convince you?

After all, we are observing various trends in computer science for years and tied with their "comes and goes" of interest in specific problems influencing work of scientists, shaping computer market and focusing effort of software developers on specific tasks. A lifetime of a trend varies form couple months to few years. Usually trend ends when something new comes up and takes over all passionate of the previous one. We can recall many big trends but some are really worth reminding: recently we went thru massive internet fascination (we still witness faze of intelligent web browsers). Now, very popular are grid computing. We still witness wave of popularity for cellular automaton and agent technique. Also fractals and chaos have their faithful fans. From time to time comes back (like a Bubonic plague) fascination of genetic algorithms and a theory of fuzzy sets.

We can say that in computer science changeability of trends is very trendy.

From the beginning of 90's neural networks started becoming popular and they are still today. A little more about this boom I wrote in preface therefore if you haven't read it yet(I know some people whose principle is to omit prefaces) I advise you to step back a few pages and read **this** one. It contains many interesting and important information that I am not going to repeat further in this book. You will find them useful if you are planning to become interested in neural networks and even use them as a useful tool in your work.

Now I am going to tell you something important about this chapter.

Well if you are not interested in biological basics of neural networks then you can skip this chapter.

Really!

In chapters ached I am going to show you how to build artificial neural networks and how you can use them. Those chapters you **must** read one after the other because if you omit something you will have problems understanding further solutions presented in this book.

But this one is different. t says the story how mankind discovered neural networks examining its own brain. Those examinations were carried out for years to find out secrets of human intelligence when suddenly research findings turned out to be useful in computer science. This chapter will tell how those borrowed from biologists artificial neural networks helps today discover another secrets of human brain. I find this incredibly interesting therefore I wrote so much on this subject. If you are eager to explore secrets of neural networks as fast as it is possible you really can skip this whole chapter. I hope that when you see how cool do they work you will come back here to learn their genesis but remember you **do not have to do it.**

If you are still reading this means that you are truly interest how did it happen to discover neural networks and I will try to satisfy your curiosity. As you already know, neural networks are simplified (therefore easier to understand and use in computer software) but surprisingly complex and interesting model of biological nervous system. Shortly we could say that neural networks are simplified model of some functions of our own brain (Fig. 1).

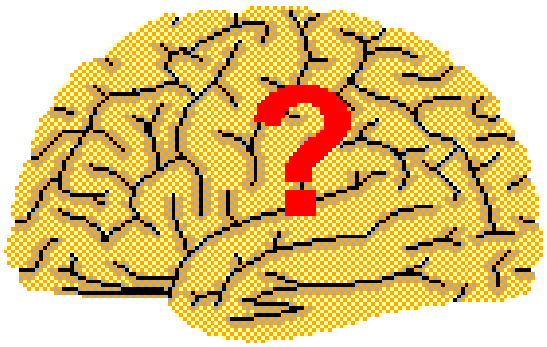


Fig. 1.1. Human brain – source of inspiration for neural networks researchers

1.2. What we have already known about the brain at the time when first artificial neural network were build?

(Translation by Stefan Turalski, stefan.turalski@gmail.com)

The brain internals have always fascinated people. However, despite of many years of intensive research, until now, we have failed to completely explain and understand a mystery of brain working.

Only during last few years we have seen essential progress in this area (we will elaborate more on this subject in the next subchapter). Whereas in the 90ties of 20th century, when the rapid development of the basis of neural networks took place, there was relatively less information on this *(brain)* subject. At that time we have already known where the most important centres responsible for the crucial motor, perception and intellectual functions are located (fig 1.2).

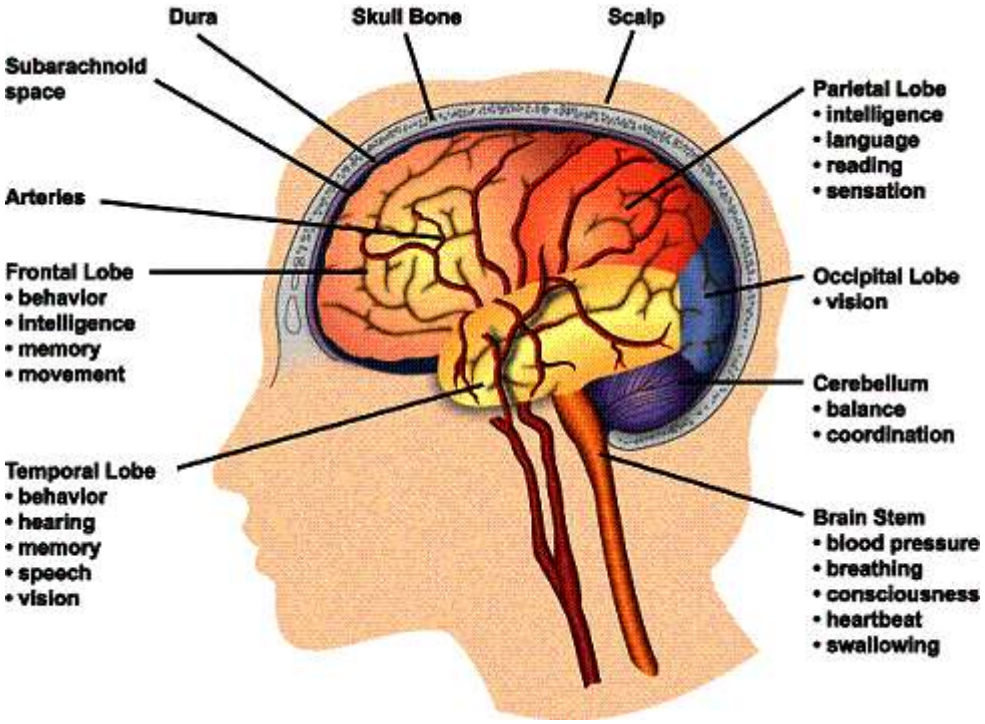


Fig. 1.2 Localization of particular functions in a brain (source of the image: http://avm.ucsf.edu/patient_info/WhatIsAnAVM/images/image015.gif)

However knowledge about particular brain elements working was rather vague. It was quite clear, how brain parts responsible for movement controlling or essential sensations (somatosensory) work (fig 1.3), because numerous diseases, injuries or war wounds let to determine, which movement (paralysis) or sensation defects are associated with injury of particular brain parts.

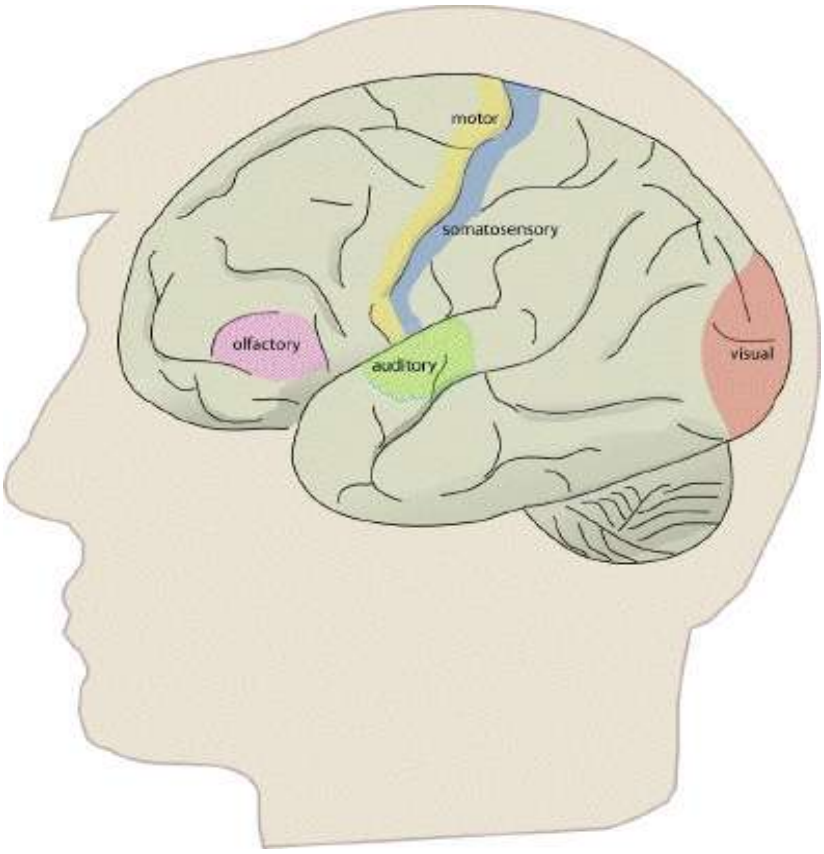


Fig. 1.3. Main localizations of brain functions (source: http://www.neurevolution.net/wp-content/uploads/primarycortex1_big.jpg)

The question about the nature and localization of more advanced psychological activities inevitably leaded, on this still quite primitive stage of knowledge. Main observation was that there seems to exist fairly well-defined specialization of individual cerebral hemispheres (fig 1.4). Main reason for that situation was a fact that ethical considerations did not allow to carry out experimental tests on human brain. The afore mentioned practice of industrious gathering and analyzing of all information regarding the relationship between functional, psychological and morphological changes that are observed in feelings and behavior of people with particular brain injury, caused in natural reason (not related to the research itself) was accepted. However a purposeful manipulation of electrode or scalpel in a tissue of healthy brain in order to gather information about the particular part functions was an entirely different matter.

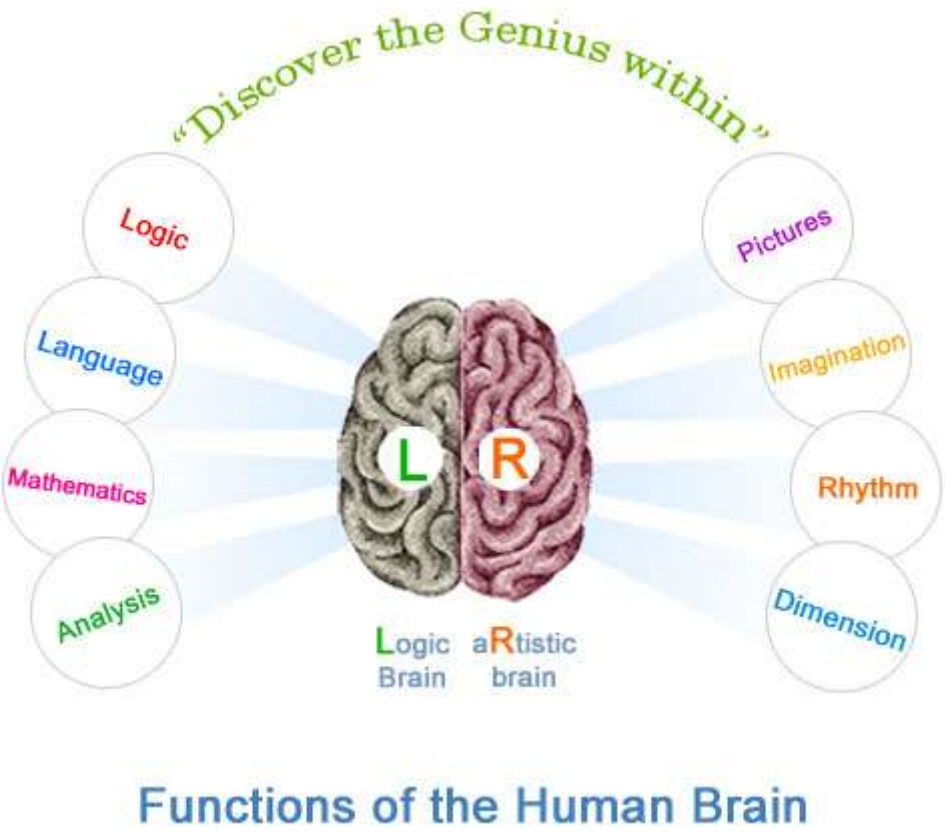


Fig. 1.4. Generalized differences between main parts of a brain (source: http://www.ucmasnepal.com/uploaded/images/ucmas_brain.jpg)

Of course, it was possible to curry out experiments on animals, and it was done very often. However, killing innocent animals in the name of our curiosity has always been rather morally ambiguous status, even if dictated by noble interest of a scientist. In addition to that, it was not possible to draw conclusions about human brain behaviors and properties directly from the animal tests, mainly because in this research area the distance between us and animals is rather more significant than in case of muscle, heart or blood researches.

What means did the pioneering neural network creators have to equip their constructions in as many features and properties modeled on the real and natural brain working as possible?

First of all, they knew that brain consists of separate cells (neurons), which are playing a role of natural processors. The first to describe human brain as a network of connected, however rather autonomous elements, was the Spanish histologist Ramón y Cajal (Nobel laureate in 1906 – see table in foreword). It was also he who introduced the concept of neurons, specialized cells that process information, receive and analyze sensations, and generate and send control signals to all these human body parts that are managed by the brain (i.e. muscles steering body movement, glands and all other internal organs). We will learn more about a structure of a neuron in the 2th chapter, because its artificial equivalent is the main component of neural network structures that are considered in that chapter. On the figure1.5, we can see how, the individual neuron was isolated from continuous web of neurons that form cerebral cortex. As it was mentioned before, this autonomous cell is a complex biological processor responsible for all functions and actions of our

nervous system (in fact, not only the brain, but also e.g. elements of sympathetic and parasympathetic nervous system that manages the activity of all internal organs).

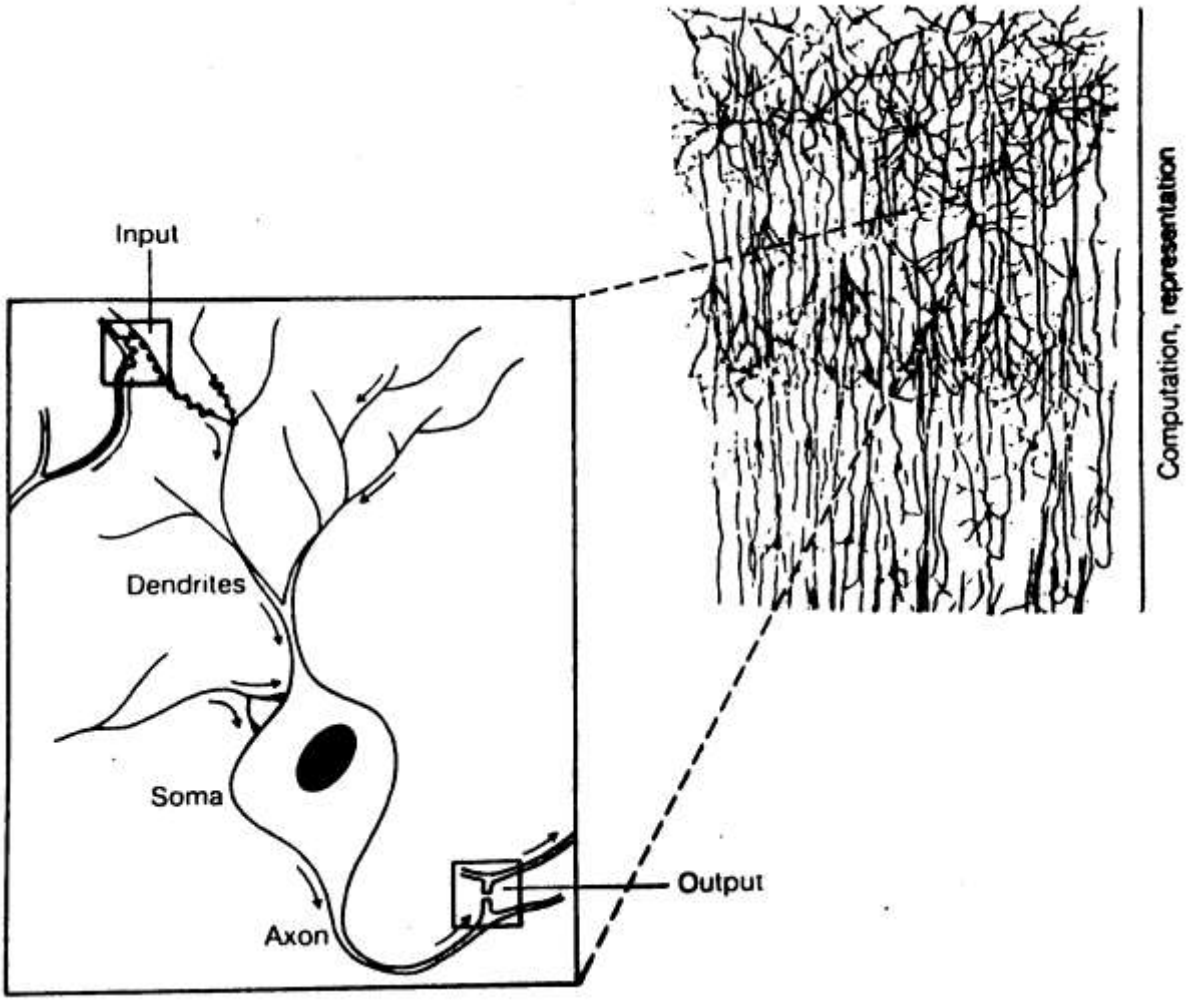


Fig. 1.5. Part of a brain cortex treated as neural network with selected neuron presentation

At the moment when first neural networks were build, the knowledge about neurons was quite distinctive, thanks to some animal species (for example *Loligio* squid*)*. These are big enough to allow clever researchers (Hodgkin and Huxley – Noble price in 1963) to find out biochemical and bioelectrical changes that happens during distribution and processing of nervous information carrier signals. However, the crucial statement was that the description of real neuron (in substance, quite complex) could be simplified significantly, by a reduction of observed information processing rules to several simple relations (these are described in the 2th chapter). Such an extremely simplified neuron (presented diagrammatically on the figure 6), still allows to create networks that have interesting and useful properties, which are at the same time very cheap to build.

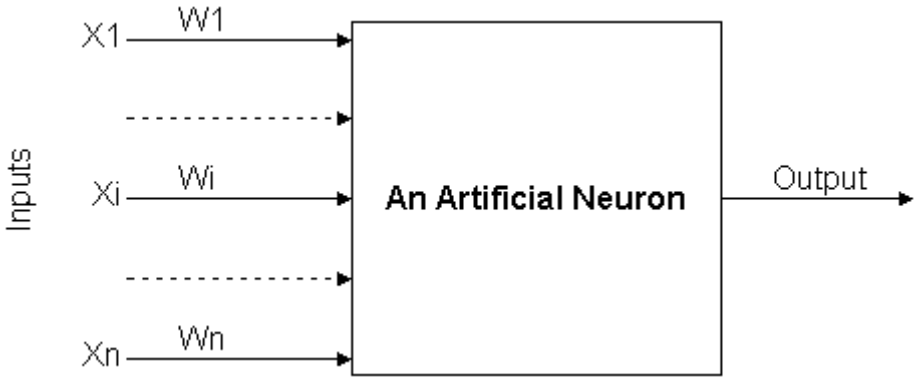


Fig. 1.6. Simplified scheme of artificial neuron

Elements recorded on the figure 1.6 (especially mysterious 'weights; marked inside the block symbolizing neuron) will be discussed in detail in the following chapters, therefore please ignore these for now. However, please note that a natural neuron has extremely reach and diverse construction (compare to the figure1.7). Its technical equivalent, presented on the figure 1.6 has substantially cut down structure and is even more greatly simplified in the area of activities that it is capable of. Despite all of that, we can, with the help of artificial neural networks), obtain such complex and interesting behaviors, as these that you will find described in further chapters of this book. It is amazing, how rich and varied are the abilities of the original, biologic network that assembles our brain!

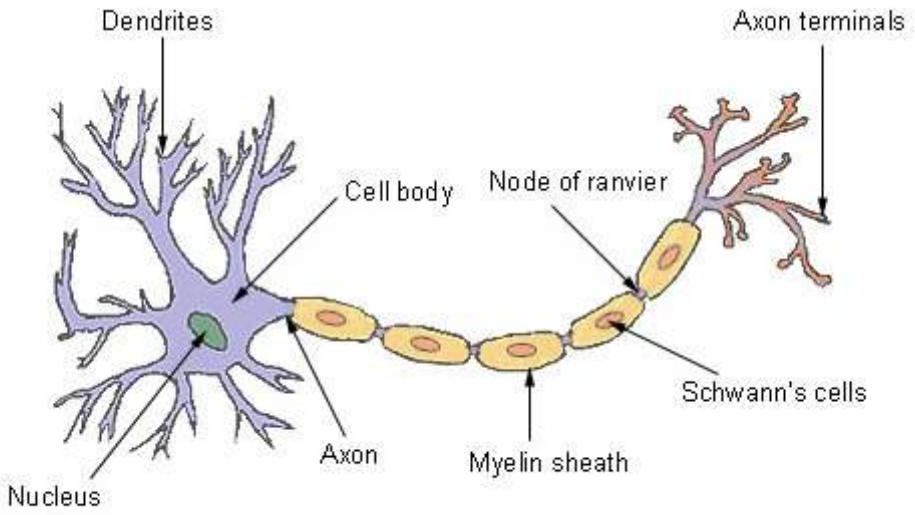


Fig. 1.7. A artistic view of biological neural cell (source: http://www.web-books.com/eLibrary/Medicine/Physiology/Nervous/neuron.jpg)

The neuron presented on the figure 1.7 is a product of a graphic fantasy, whereas on the figure 1.8 you can see an example of a real neural cell dissected free from a rat brain – human neurons look almost identically. Such a biological neuron has really complex structure, hasn't it?

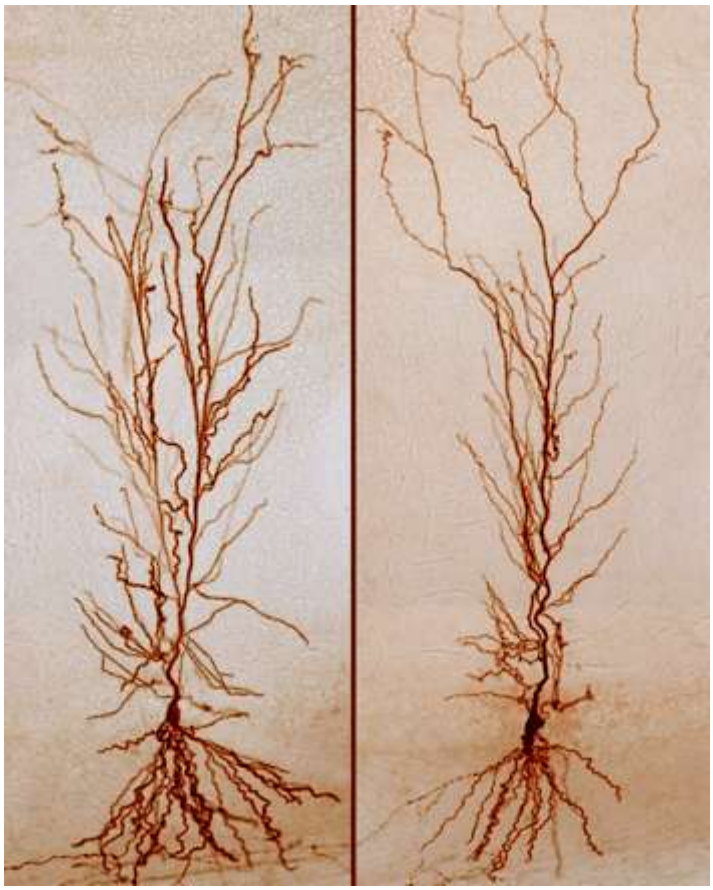


Fig. 1.8. Real view of the biological neurons (from the brain of rat) (source: http://flutuante.files.wordpress.com/2009/08/rat-neuron.png)

Because of the fact that artificial neurons are so simplified, it is possible to realize them technically in an easy and cheap form of uncomplicated electronic system (first neural networks were built as specialized electronic machines called *perceptrons*). It is also fairy easy to model these in a form of algorithm that simulates an activity of such a cell using standard PC system (or other computer type). In currently utilized systems the simulation realization is chosen practically as a rule. It constitutes a convenient and cheap tool, which allows simulation of single neurons, as well as all networks of these.

1.3. How were the first neural networks built?

(translation by Agata Krawcewicz, hogcia@gmail.com)

Let's get back to computing the biological information, on which the creators of the first neural networks based on, to show what neurocybernetics did with these biological information in order to make the obtained networks cheap and comfortably used. They already had considerable knowledge about **what actions** can the biological neuron do. It is enough to look into table P.1 in the preface, to see how many Nobel prizes had been awarded throughout the whole XX century for discoveries, which - indirectly or directly - concerned the very problem of neural cell and it's functioning. The most important information, which the biologists managed to find, was related to places, in which one neuron passes a signal to another neuron (Fig. 1.9). It was astonishing and fascinating. They

found out that during processing information in the brain, the large an complicated cell bodies or the long bushy neural fibers (*axons* and *dendrites*), which are used for communication between neurons, are not the most important thing. What was significant, were the systems that mediate in the process of passing the information between neurons, called *synapses*. They are very small, in fact so small, that the resolving power of the optical microscopes, typically used in biology, was too small to find those structures and describe them. You can also barely see them on picture 1.9.

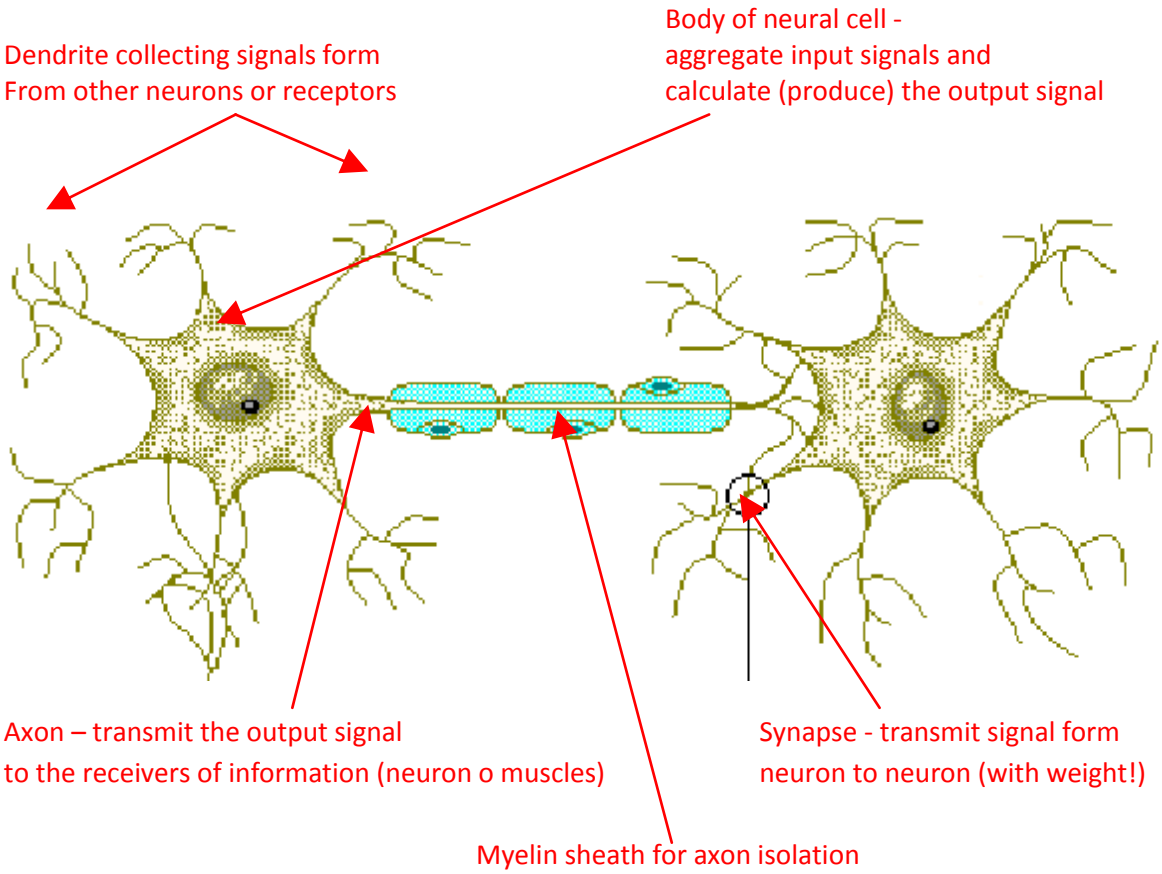


Fig. 1.9. The smallest functional part of neural system: two connected and cooperating neurons. In this structure most important part is synapse connecting such two neurons.

After the electron microscopes discovery we could show how complex and interesting shapes do the synapses really have (Fig. 1.10), and a series of brilliant experiments of a British neurophysiologist, John Eccles proved, that when a neural signal goes through a synapse, a special chemical substances are engaged - the so called neuromediators, which are released at the end of the axon from the neuron that transmits the information, and travel to a so called postsynaptic membrane, which is contained in the neuron that receives the information (see the greatly simplified schema of a synapse structure, which is shown on picture 1.11). In the greatest simplification we could say, that teaching a neuron (and the whole brain) depends on the fact, that the same signal sent through an axon from the cell that transmits the information can release a greater or smaller quantity of the neuromediator to the synapse that receives the signal. If – in the process of learning – the brain finds out that the signal is important – then the quantity of the neuromediator is increased. If it's settled that the signal is not important, the quantity of the neuromediator is decreased. That is how the mystery of teaching and memory looks like, though I want to stress it once more – the mechanism

sketched in the previous instant is extremely simplified in relation to the complex biological processes, which in reality take place in a synapse. The very best proof for how important and difficult process do we have here, is the fact that for discovering how the synapses transmit information from a neuron to neuron, and also for disclosing the mechanism of the changes that take place in synapses, when a brain is learning and acquiring new information, John Eccles was awarded a Nobel prize in the year 1963 (see the table contained in the preface).

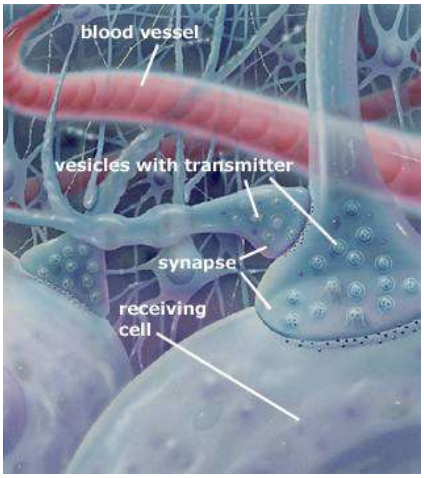


Fig. 1.10. View of the synapse. Reconstructed on the basis of hundred observations performed by means of electronic microscope. (source: http://www.lionden.com/graphics/AP/synapse.jpg)

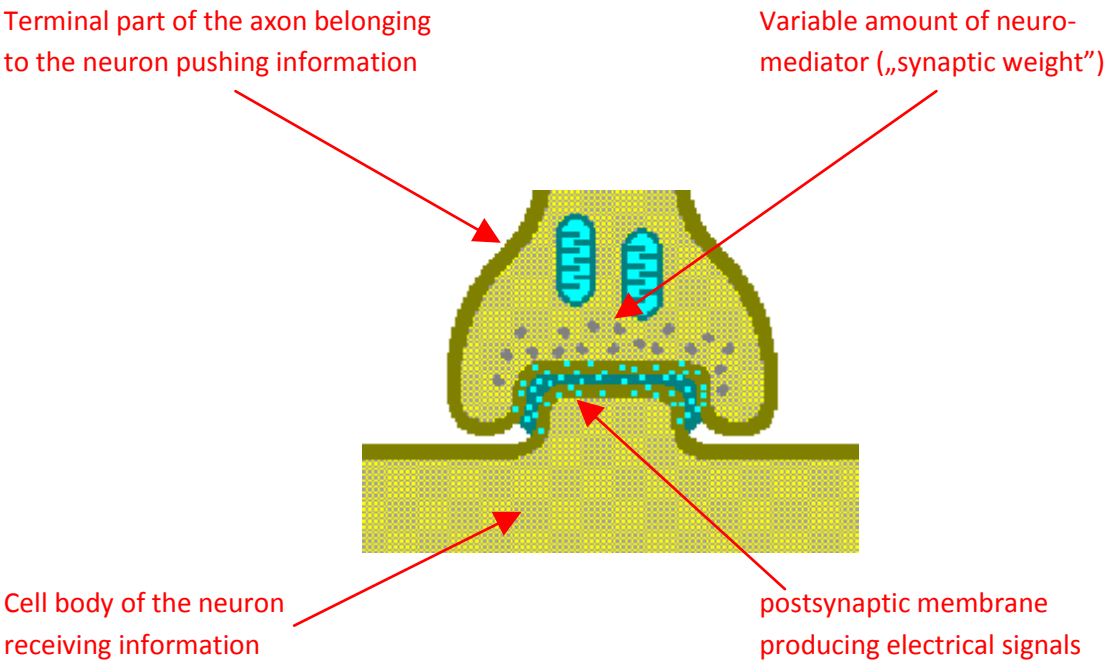


Fig. 1.11. Very simplified scheme of the synapse structure and its main elements

The neural networks specialists readily used the information and thanks to that the systems built by them have profited by one of their most important attributes: **a capability to learn**. Of course again the biological rule of learning, which is objectively rather quite complicated and appeals to very

complex biochemical processes (Fig. 1.12) had to be greatly simplified, so that it would be possible to obtain a tool, which could be efficiently used as a system for resolving practical computer science problems. Furthermore, it was decided, that the subject of learning in neural networks would be solely the kind of knowledge, which is classified by psychology as the **procedural memory**, which though, as we know, is not the only type of memory that a human being has (Fig. 1.13).

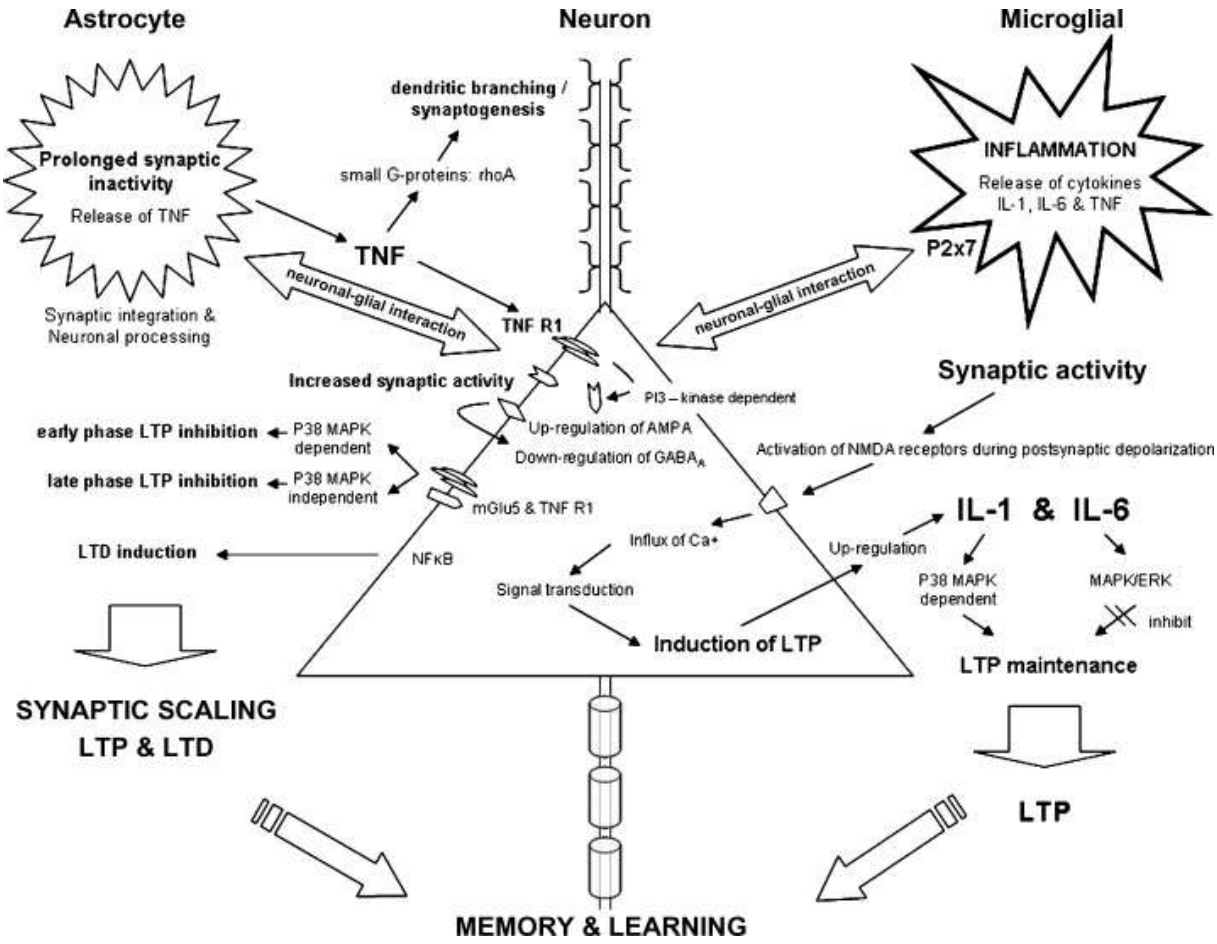


Fig. 1.12. Biochemical mechanisms of learning and memory (source: http://ars.els-cdn.com/content/image/1-s2.0-S0149763408001838-gr3.jpg)

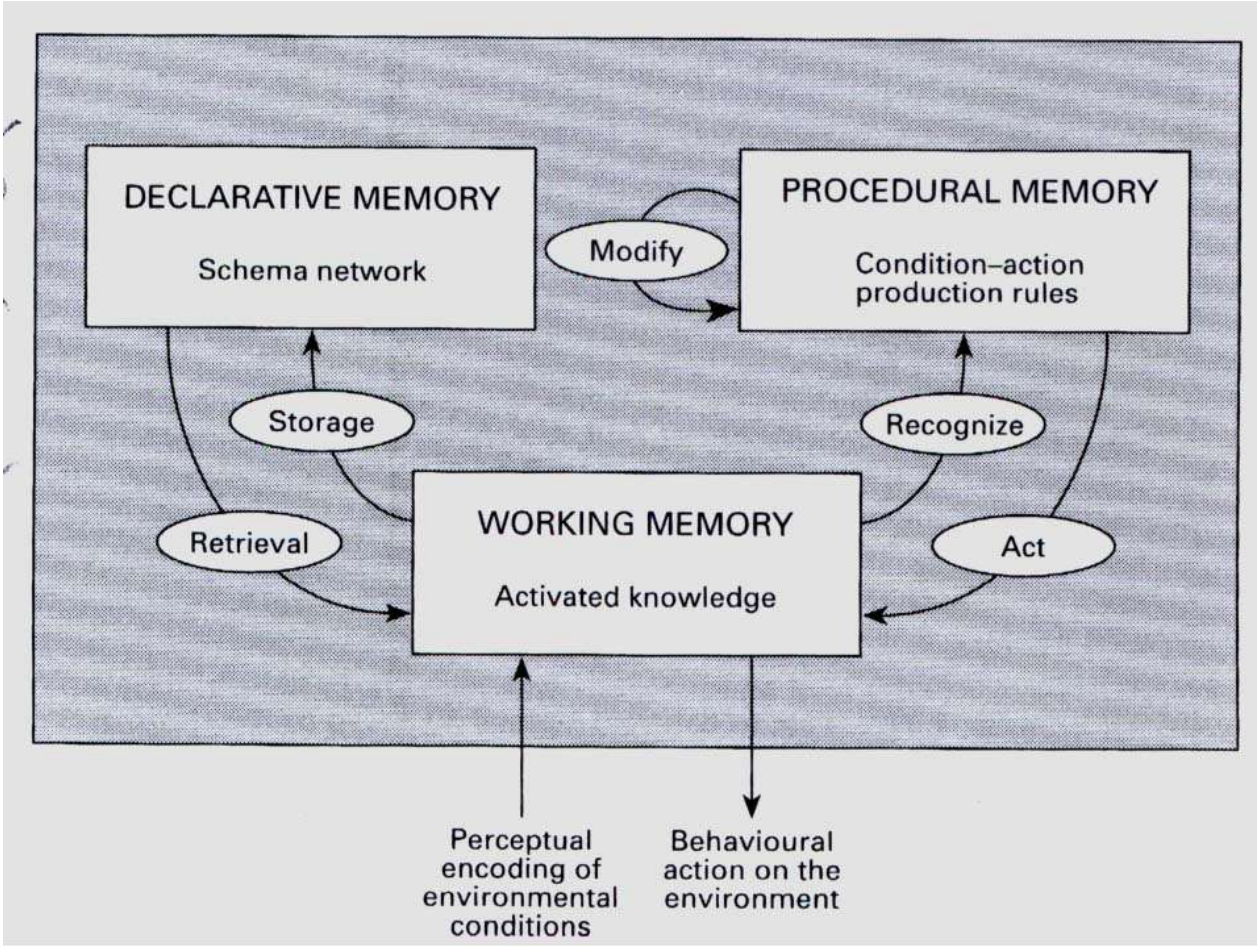


Fig. 1.13. Different types of memory in human cognitive processes

The third source of information, on which the concept of neural networks was based in the 90ties, was that the inner structure of brain was identified at that time. Persistent work of many generations of histologists, analyzing thousands of microscopic specimens and hundreds of less or more successful attempts to reconstruct the three-dimensional structure of the connections between neural elements bearded such a fruit, that schemas were available at that time, such as the one exemplary presented in figure 1.14.

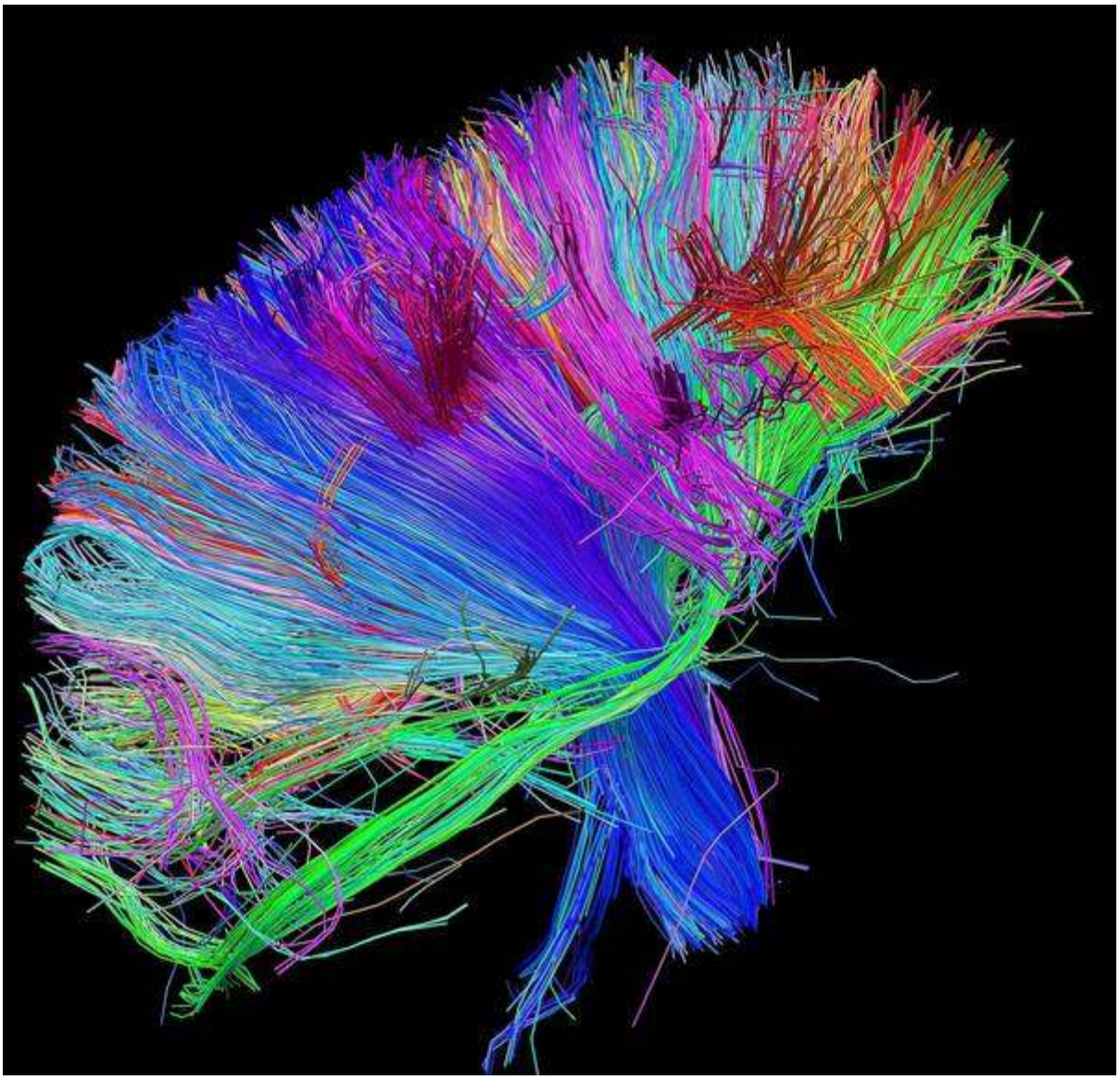


Fig. 1.14. Three-dimensional internal structure of the connections in the brain (source: http://www.trbimg.com/img-504f806f/turbine/la-he-brainresearch18-154.jpg-20120910/600)

1.4. Why should neural networks consist of layers?

(translation by Krzysztof Królczyk, scoorviel@go2.pl)

Similarly to earlier mentioned simplification of biological information (about other real brain properties), this also applies to space layout of neurons, and connections they create – the whole science complexity, neuroanatomic and cytological knowledge were reduced to absolute minimum. Neural net designers focused heavily on implementing working model - practical, tough extremely truncated. It appears, we could observe regular pattern, which neurons tend to create in several brain areas. Below, we can see few examples of such layer-like structure (Fig. 1.15).

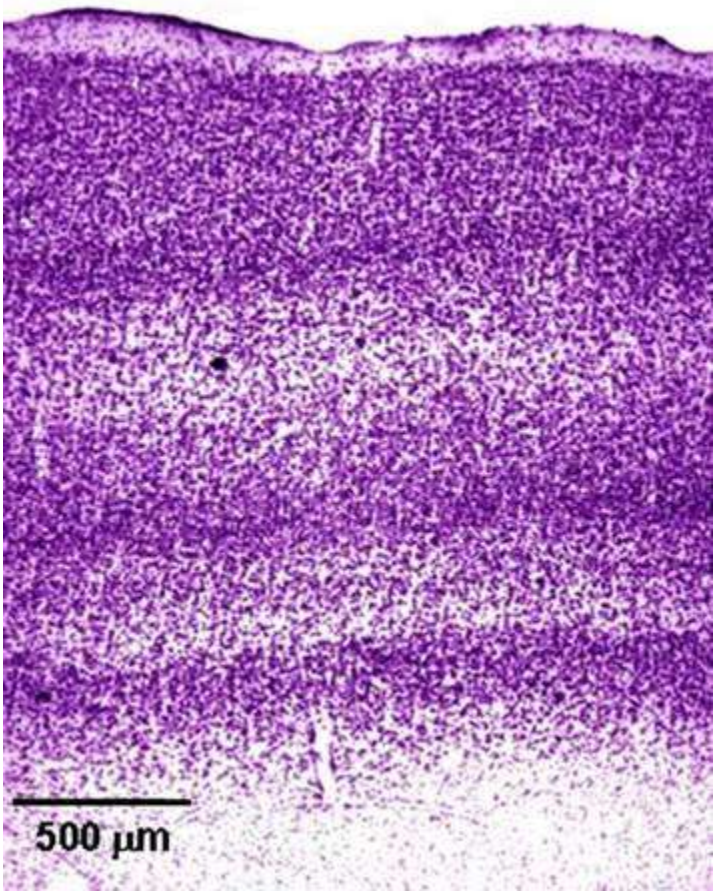


Fig. 1.15. Layered structure of the human brain cortex (source: http://hirnforschung.kyb.mpg.de/uploads/pics/EN_M2_clip_image002_01.jpg)

Retina is another example of such structure (Fig. 1.16); as for embryologists – being transformed part of cerebral cortex.

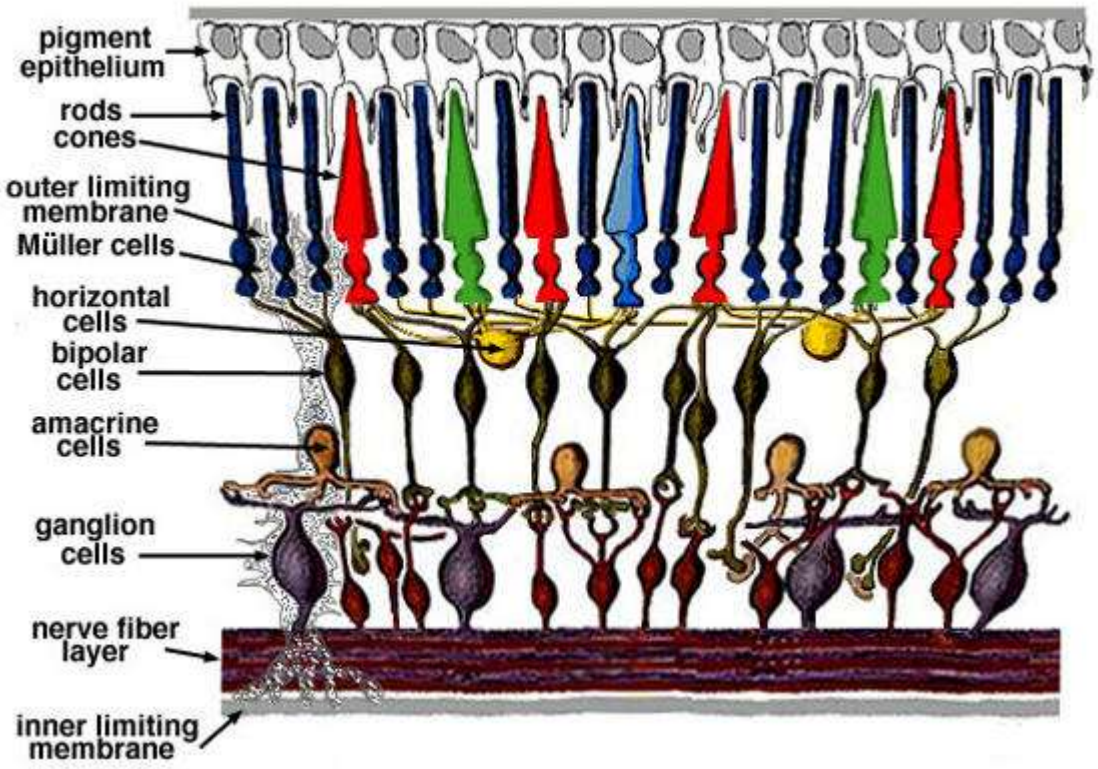


Fig. 1.16. Retina (part of the eye) is also organized as layered structure (source: http://www.theness.com/images/blogimages/retina.jpeg)

It's safe to imply, that neural networks, designed as a multilayer structure are quite convenient - technically it's the easiest way; however, neural nets are biologically "crippled" models of actual tissue, nevertheless functional enough to assume that results obtained are fairly correct – at last in context of neurophysiology. According to words of one green ogre "Ogres have layers. Like onions.". Neural networks have layers also.

Typical neural network, therefore, has structure shown in Fig. 1.17.

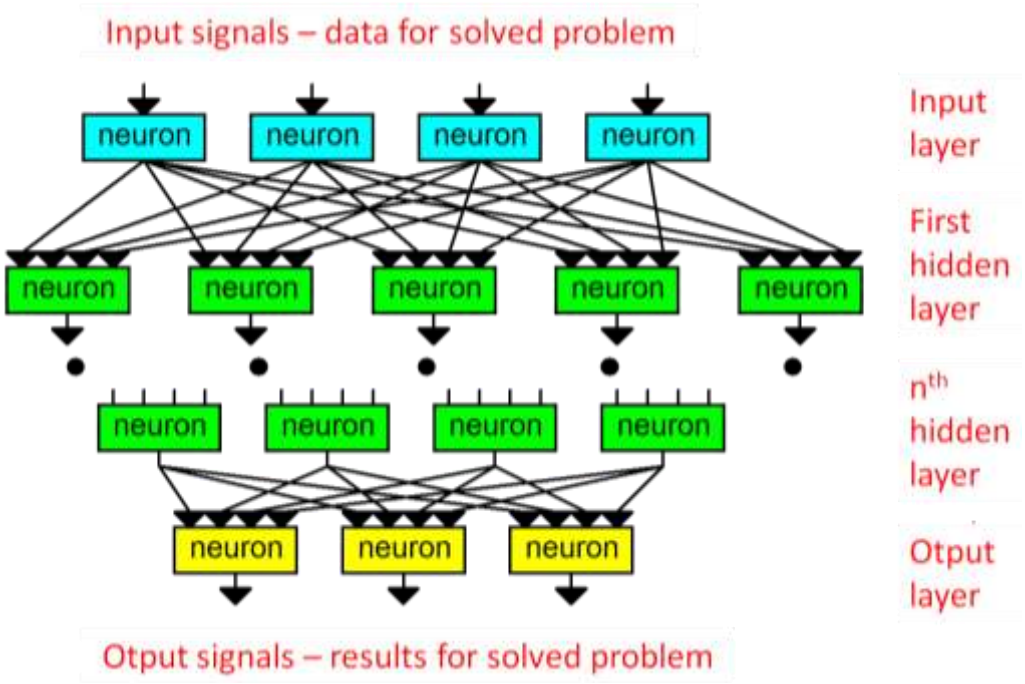


Fig. 1.17. Typical neural network structure

What are main advantages of such (layer) approach? It's very simple to create a model, and simulate it's behavior using various computer programs. That's why researchers adopted such structure and use it from there on, in every neural net. Let's say it again – it's very inaccurate if considered as a biological model, however main idea is preserved. There were, of course, remarks, how much better would have networks operate if model closer resembled its origin, real tissue, or how could it be adjusted to perform specific tasks, but as for now, none worries about it.

Another problem is with connecting layers. For example, in real brain, schematic of neural connections is quite complicated, and differs depending on which brain area is considered. Therefore, in XIX century, first topological brain map was created, dividing it in view of identical neural connections templates. (Fig. 1.18).

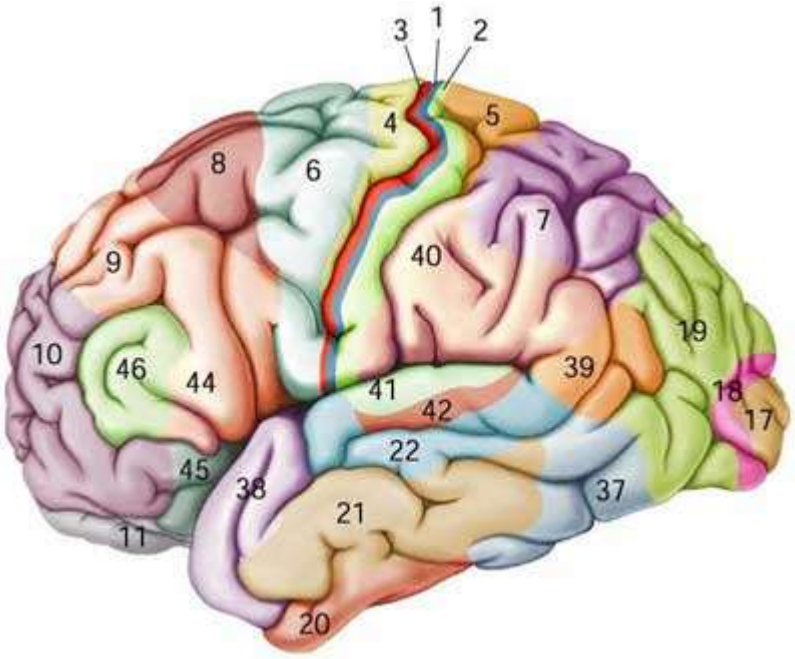


Fig. 1.18. Map of regions with different cell connections in the brain (by K. Brodmann) (source: http://thebrain.mcgill.ca/flash/capsules/images/outil_jaune05_img02.jpg)

Here, with same color, are marked fragments with microscopically examined similar connections, whereas different colors corresponds to substantial differences. This map, with rather historical meaning, was called Brodmann's areas. Brodmann divided the cortex into 52 regions. Currently we treat brain much more subtle; however – this is a good example of problem we're facing when analyzing *"how are neurons connected into net",* the answer varies with different brain part.

If we were thinking about the sole purpose of building artificial neural networks – one may think it's essential to adapt its connection structure to single problem we're dealing with. That's true, it's been proven that well chosen structure can greatly increase speed of net's learning. The problem is situated, unfortunately, elsewhere. In most problems we are trying to solve – we can't really tell what is best way to work the problem out. If we can't even guess which algorithm is suitable, and which one is network going to employ after learning process, the less could we be capable of selecting (a priori), network elements which are necessary, from useless.

Therefore – the decision about connecting layers, and single elements in networks are arbitrary, and usually it's full connection – each element is connected to all other. Once again – such idea of homogeneous, full connection schematic – reduces effort required to define network, however increases computing complexity, i.e. higher memory usage, or chip complexity, needed to recreate all connections between elements. It's worth noting, that without such simplification, network definition would require thousands parameters, surely causing employment such structure, to be a programmer's worst nightmare. Whereas using fully connected elements, again, is basic thoughtless designer's will. It's almost a practice; also, causes no real harm, since learning process eliminates unnecessary connections from whole bunch.

1.5. How far from the biological brain was the first artificial neural network?

(Translation by Arek Janeczko, ajaneczko1@gmail.com)

Summing up, it is necessary to state that Artificial Neural Networks, which came into existence in the 90ties of the 20th century, had strong foundations in anatomical, physiological and biochemical knowledge about the human brain, which at that time was available. However, authors of neural networks did not attempt to make an exact copy of this knowledge and rather treated it as an inspiration. Therefore, the construction and principles governing artificial neural networks applied in practice are not the exact reflection of the biological knowledge, even that outdated one, from the past a dozen or so years. Basically, we can say that the workshop of the inventor of modern neural network consists to a certain degree of biological knowledge elements, which could be easily discovered in the background and are rather the source of inspiration than the precise pattern on which his actions are based. Nevertheless the matter which is shaped by neuro-cybernetic scientist is totally from the field of Computer Science, because neural networks are created, learn, are the objects of research and are applied in typical present computers. A graphical metaphor of this process could be assembled by the AGH students the image presenting the author of this book as the explorer of neural networks (fig. 1.19).

Fig. 1.19. The knowledge about biological neurons “soaking” into the structure of presently used computers (the picture of the author of the book made by the AGH students)

Therefore in all further considerations we will remember (and we will be taking into account) that during forming neural networks all biological knowledge that we have at first has been deeply thoroughly looked through and simplified and then it has been used as the basis for constructing artificial neurocybernetic systems. This fact fundamentally influences the properties of examined neural networks which in a consequence are neural more from its name than because of the real

similarity to the actual brain. Contrary to appearances this fact positively decides about the properties and the possibilities of neural networks – which are considered also as tools assisting in the process of biological understanding of our own brain. We will address this issue in the next subsection of the book.

While preparing the next edition of the book which would appear in print over ten years after its first edition, I could not ignore the fact that during this decade the research over the brain has considerably gone forward, and the concepts on the subject when and for what purpose neural networks should be applied for, also have undergone a certain evolution. Because of this in the next subsection, which was not present in the earlier edition, we will talk about concurrent problems connected with human nervous system research and I will try to show you what role neural networks might play in these researches.

1.6. What methods do we currently use in brain research?

(Translation by Krzysztof Królczyk, scoorviel@go2.pl)

Modern wide variety of „tools" available for human mind researchers in XXI century, is simply incomparable, with those used by pioneers who gave us base knowledge about structure and functionality of nervous system. Their work underlined creation of first neural network in 90-ties of last century. To begin with, let's name several, which led us to improved representation of human brain, and it's internal structures. Without computer tomography (CT) and nuclear magnetic resonance (NMR), skull's shell, or precisely cranium, was impassable for more primitive observation techniques (including x-raying), enviously hiding secrets of healthy(and living!) human brain. We were able to see into it only after patient's – and concurrent – brain's death. What, of course couldn't give us any insight of its activities. Anatomists, happily cutting brain into pieces, staining and watching it under microscope, tried to work out core structures processing information, and it's connectivities – more or less with same luck, as if someone were cutting computer to pieces, trying to find out how were wires connected, and what happens inside integrated circuits, thinking he's learning informatics by doing so... Internal brain structures of living man can be shown, and analyzed today with an extraordinary precision – take a look at Fig. 1.20.

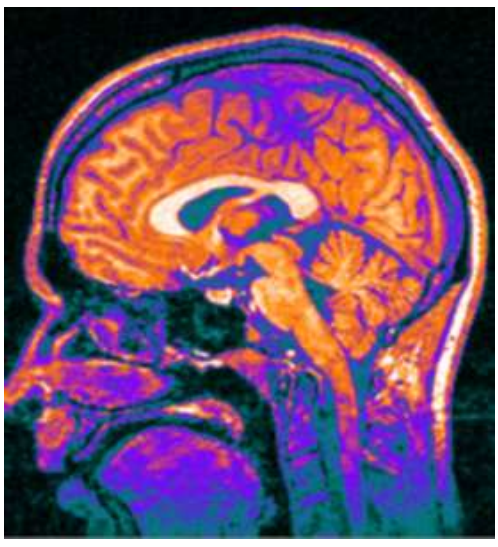


Fig. 1.20. Internal brain structures of living man can be analyzed today with astounding precision... (source: http://www.ucl.ac.uk/news/news-articles/1012/s4_cropped.jpg)

Diagnostics allow us to detect and locate certain places, showing high brain activity at current moment – have a look at Fig. 1.21. Linking such sections, with type of activity a person is performing at the moment, we could presume, that certain brain structures are respondent for certain tasks. Thus led us to better understanding of functional aspects of neural compounds. Let's have a look at example below.

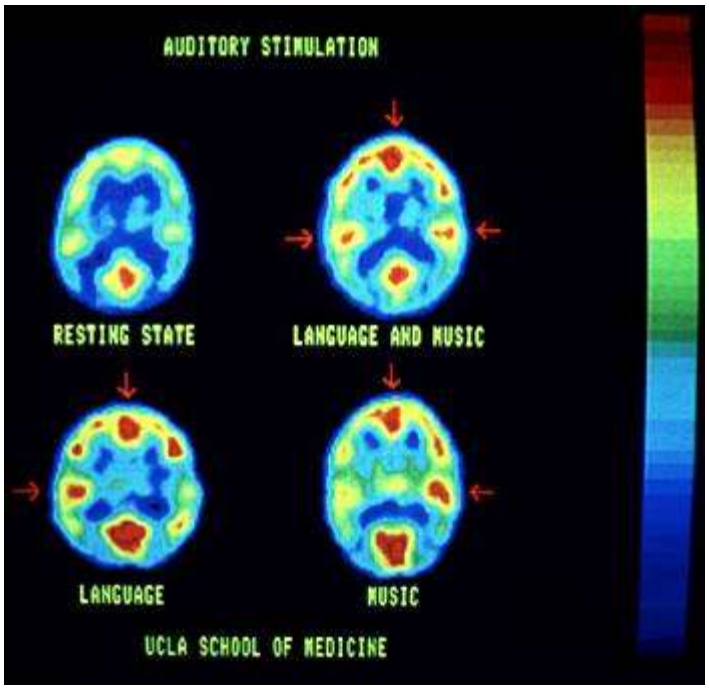


Fig. 1.21. Method showing brain activity, during performance of certain tasks by research subject (source: http://faculty.vassar.edu/abbaird/resources/brain_science/images/pet_image.jpg)

Look closely at Fig. 1.22. It shows an image (consisting of four profiles at different height, obtained by PET - Positron Emission Tomography) of human brain viewed from inside. It's owner is awake, however not focused on anything particular, given no task to accomplish. We can see, that however most of his brain is inactive (shown as blue or green areas), there are places where neurons work (shown as yellow, and red areas) Why's that? We should remember - that our object, although relaxed, thinks! He's still moving, even if unnoticeable, feels cold or heat etc. This activity is rather scattered and weak.

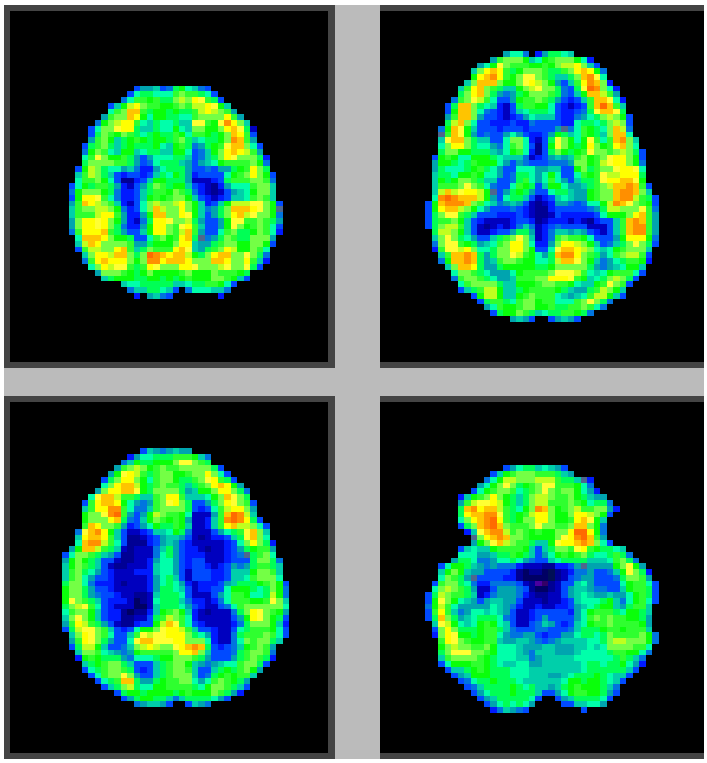


Fig. 1.22. A still image of human brain, whose owner isn't focused on anything particular.

Of course, if patient focuses, i.e. on solving difficult mathematical problem, parts responsible for abstractive thinking immediately increase their activity, resulting in red areas at Fig. 1.23.

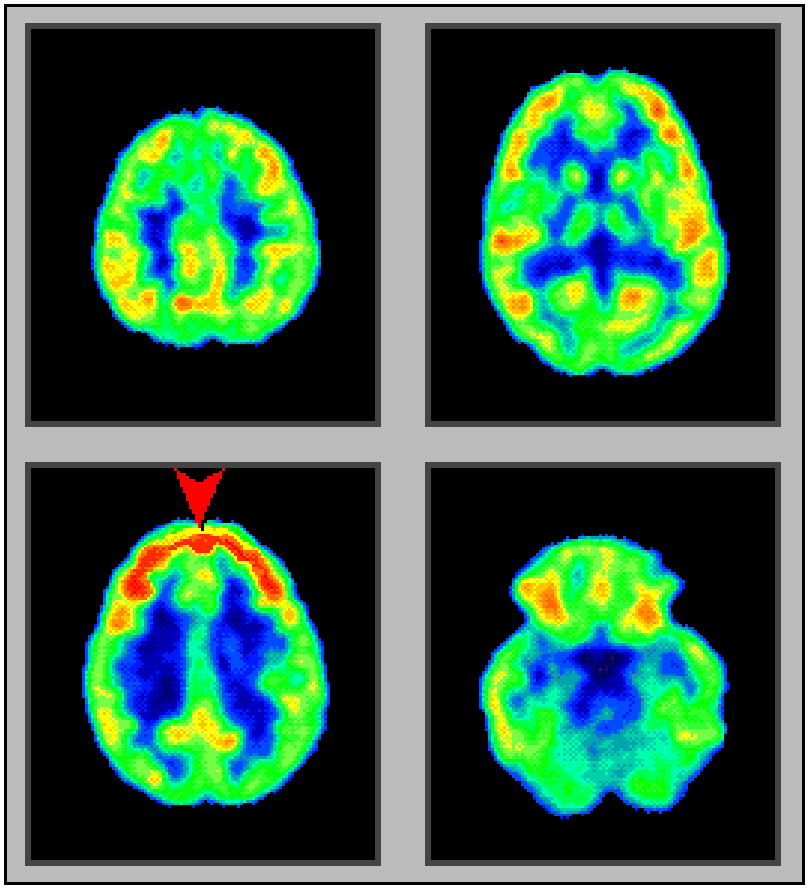


Fig. 1.23. An image of human brain, whose owner focused heavily on mathematical equation. Intensive red marks frontal lobe, activated by this exercise

This method can be used not only to study aware exhausting efforts. If some picture suddenly draw attention of our research object, in his rear brain parts millions of neurons used to analysis, responsible for perception and recognition of visual signals (as shown at Fig. 1.24)

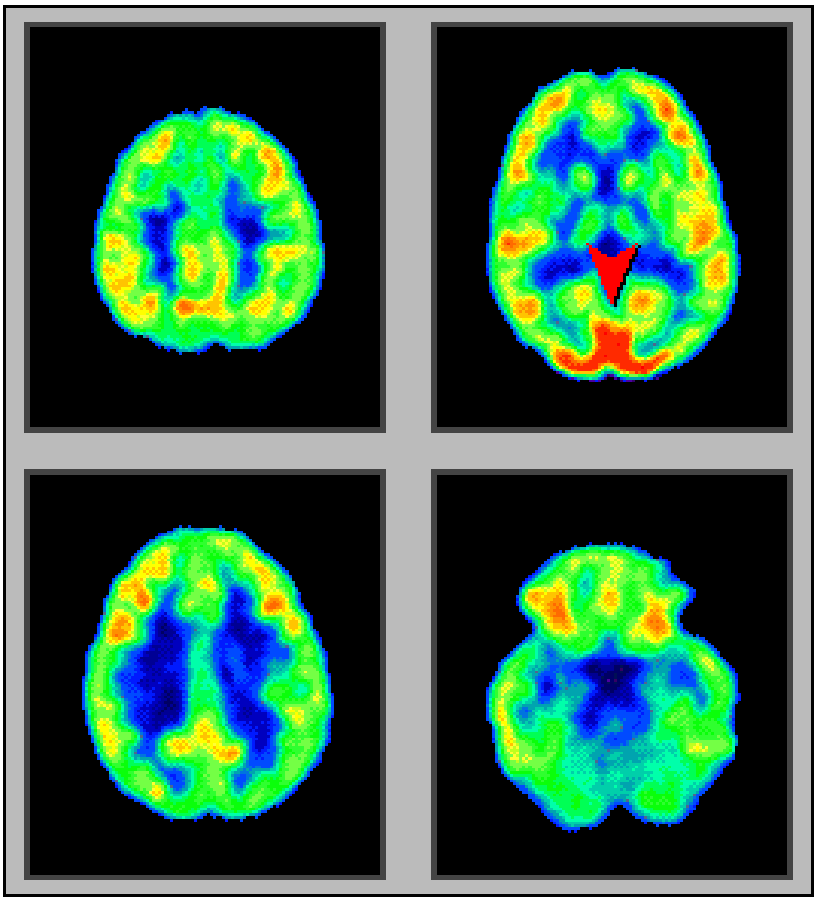


Fig. 1.24. Man's brain activity during watching something interesting. We can see red spots marking activity in rear lobes (arrow) responsible for acquisition and identification of visual signals

Accordingly during listening and speech comprehension – hives responsible for analysis and remembering sounds, are activated in temporal lobes, what shows Fig. 1.25.

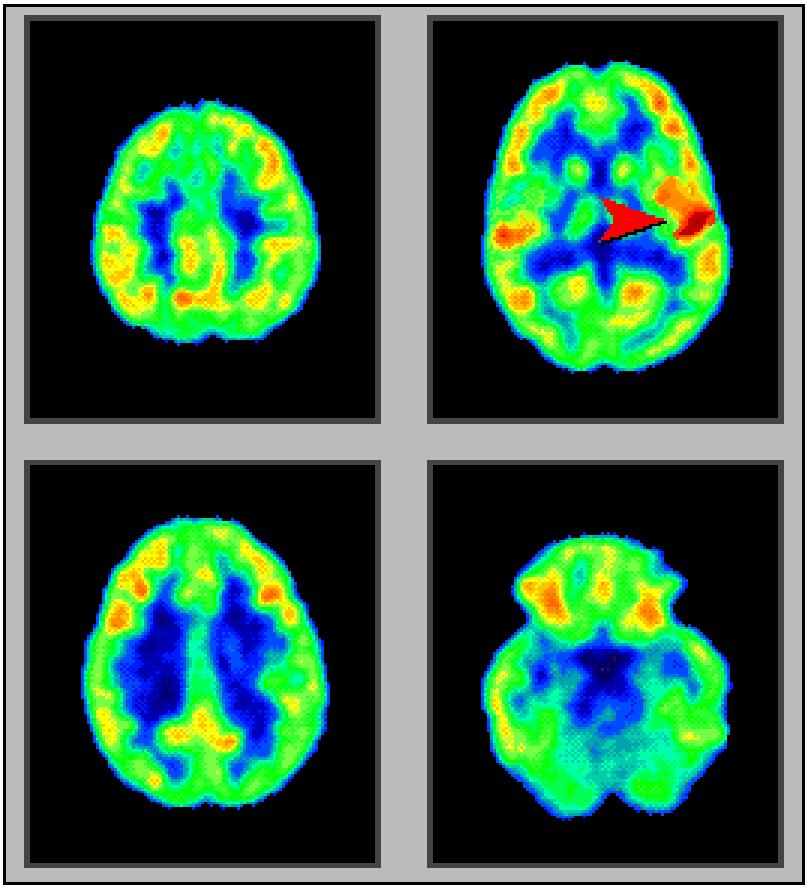


Fig. 1.25. A man listening to a conversation activates mostly temporal lobes since they are responsible for signal analysis. It's worth noticing, that hence speech comprehension areas are located only in one side of brain; if an object were listening to music – both image sides would be red.

Methods, which were shortly described here, allow us to record not only temporary states, or explaining how is brain organized; in fact – developed **combined** techniques of imaging internal structures, capturing it's partial activities, gave us insight about correlation between certain brain areas and patient's corresponding actions (Fig. 1.26).

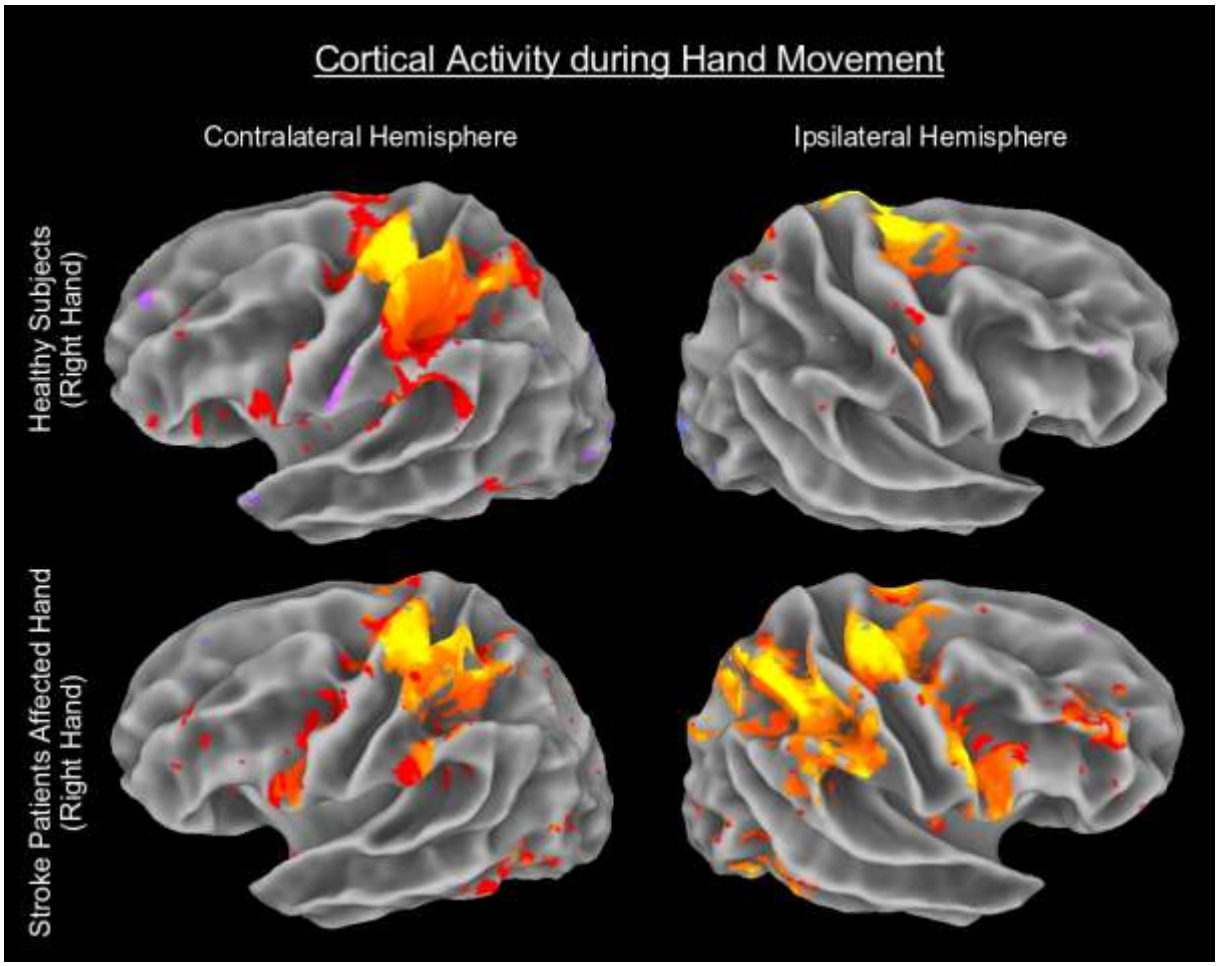


Fig. 1.26. Tracking changes connected with performing specific tasks. (source: http://www.martinos.org/neurorecovery/images/fMRI_labeled.png)

These changes can be recorded **dynamically** (creating something like a movie, or computer animation) what is enormous advantage over static pictures, or any other research results. I'm sure You've seen pictures of a runner, and his motion divided into sections, or concurrent snapshots of animated movie. At Fig. 1.27, and subsequent, we have example - animation of such brain (anatomical) structures. Presenting such still images one after another in timely fashion, gives us illusion of fluent movement so important to freely track which structures activate, it's order, during analysis of particular action.

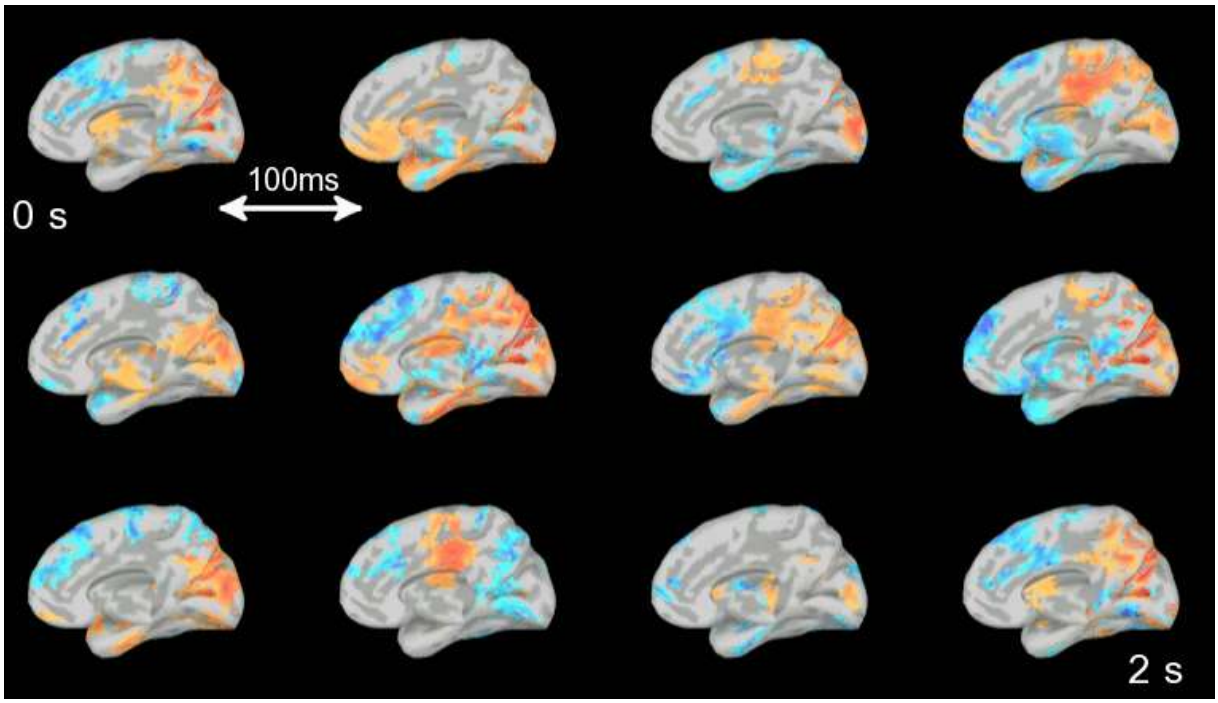


Fig. 1.27. Presentation of dynamic processes happening inside brain. (source: http://www.bic.mni.mcgill.ca/uploads/ResearchLabsNeuroSPEED/restingstatedemo.gif)

1.7. Do the neural networks can help in studies on the mystery of the human mind?

(translation by Natalia Kubera, natalia.kubera@googlemail.com)

The **examples** presented above were obtained by modern methods of brain examination. They seems to show that after the 1990s, which were called „the decade of brain", because of the intensity of research, we know so much about the human brain, that it is no longer a mystery.

This belief is misleading. Currently, the brain structure is quite well explored. We have also some knowledge about its functionality. However, a common problem for studies on highly complex objects appeared: the reconstruction of a whole on the basis of waste amount of separate, distributed, particular information. One of the very efficient methods used in modern science, is the method of decomposition – in other words – the method of division into parts. Do you have a problem with description of a huge system? Divide it into hundred pieces and investigate each of them! You cannot understand a complex process? Let's find several dozen of simple sub-processes that go into making of this complicated one and investigate each of them separately. This method is very effective. If separated subsystems or sub-processes are still resistant towards your scientific methods, you can always divide them into smaller and simpler parts. However, it causes one, characteristic difficulty: Who, when and how will compile the results of those numerous researches of all the fragments?

While analysis of really complex systems, the synthesis of particular results is not easy, especially if those results were obtained by various scientific techniques, which are characteristic for each, applied science discipline. It is hard to find a person, who would be able to integrate, for example: based mainly on drawings anatomic information with based on case descriptions physiological data and results of biochemical marks obtained by analytical apparatus. Computer modeling can work as a

kid of a „common denominator„ for the all, obtained from various sources information, so if it is only possible, it is worth to use it. It was experimentally proved that computer model describing anatomy **can be joined** with computer record of physiological processes and computer description of biochemical reactions. The current approach attempts to rebuild our entire knowledge on various, investigated separately by different scientists biological systems.

With reference to neural system and in particular to human brain, various computer models may be used in order to combine results of many different researches, which can give a chance to understand an **integrated** functionality of this extraordinarily complex system. Some of those models will be mentioned further in the text. Whereas in this preliminary chapter, I would like to state that the easiest approach is to attempt to describe our knowledge about brain in reference to considered in this book, artificial neural networks. Obviously, the human brain is very complex and much larger than **very** simplified neural networks. However, it is common in science that by usage of simplified models we discover rules which also come true in greater scale. For example, if a chemist carries out a reaction in a small sample, we are entitled to presume, that the some reaction takes place in a vast ocean or inside a distant star – and usually it is true. Moreover , it is much easier to handle a small sample that a vast ocean and all the more a distant star. While looking for a scientific truth, the **simplification** is often a key to success.

That is the reason why, despite of neural networks simples (which is close to primitiveness!), more and more frequently, we hear about researches, which use artificial neural networks for modeling human brain processes in order to obtain a basis for its deeper comprehension. In order to realize how much we can rely on this tool, let's try to compare the complexity of our artificial neural networks with animal and human neural systems.

1.8. How much artificial neural networks are simplified in comparison to biological ones?

(Translation by Ryszard Tadeusiewicz; rtad@agh.edu.pl)

Neural networks are of f course very simplified in comparison to real neural systems of most living creatures. It can be observed on Fig. 1.28, where region occupied by typical neural networks (realized as usually as programs for general purpose computers) marked as yellow square in coordinates showing on abscissa structural complexity of the considered neuroinformatic system and on ordinate speed of the system functioning. Both dimension on this plot are represented in logarithmic scale because of huge distance between smallest and biggest presented values. For example structural complexity measured by number of synapses in considered neuroinformatic system can vary from 10^2 for typical artificial neural network used for technological purposes up to 10^{12} for the human brain. This dimension for artificial neural networks is limited by value about 10^5 – 10^6 because of computer memory limitations, where appropriate values for not very complicated “brains” of fly or bee can be characterized by numbers of synapses 10^8 – 10^9 respectively.

In comparison with these neural systems brains of mammals are really huge with 10^{11} synapses for rat and 10^{12} synapses for human central nervous system.

Let consider almost linear relation between structural complexity of such (taken into account) biological neural systems and speed of the their functioning (Fig. 1.28). In fact it is general rule, caused by massively parallel method of biological neural systems functioning. For this type functioning when system consists of more elements (more neuron and more synapses) and all these elements working together (simultaneously) – speed of data processing increases proportionally to the system structural dimension.

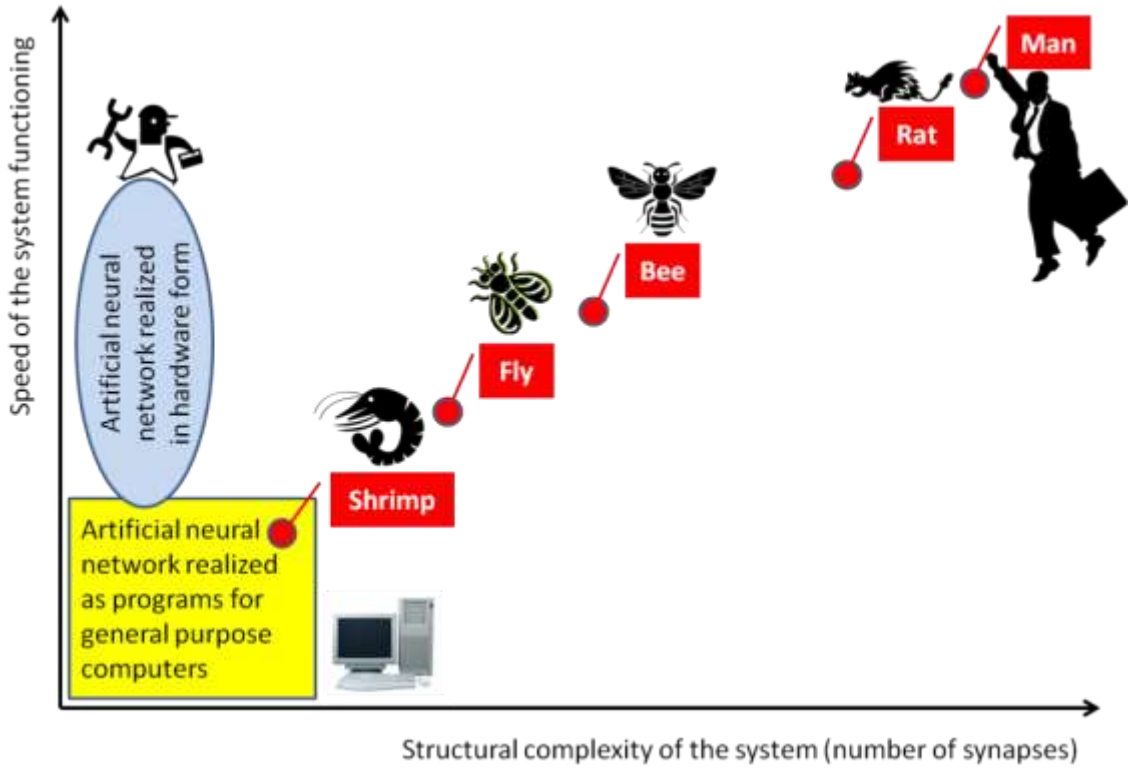


Fig. 1.28. Localization of artificial neural networks and selected real neural systems on diagram showing relation between Structural complexity of the system (number of synapses) and speed of the system functioning

For artificial neural networks speed of system functioning depend on the form of network realization. When neural network is realized as a program simulating neural activity, learning, problem solving etc. on general purpose computers (including laptops, tablets and palmtop devices) – the functioning speed is limited by performance of used processor. It is evident, that is impossible to speed up processing time over hardware limitations using any type of programs, therefore artificial neural networks realized as a programs on general purpose computers are rather slow.

Is possible achieve very fast functioning of artificial neural networks when there are realized in hardware form (see blue ellipse on Fig. 1.28). In bibliography or in internet you can find many examples of neural networks realized as specialized electronic chips – recently often in FPGA technology. Are known also optoelectronics solutions, chips fabricated using partially analog technologies (most systems are of course digital taking into account input , output and general control of the system , but sometimes smart analog devices inside can be incredible fast). Known are also neurochips made from both electronic silicon part and biological part – living neural call or neural tissue piece treated as a device component.

All such methods of hardware realization of artificial neural networks can be very fast (see Fig. 1.28) but the structural complexity of such systems is always very limited and elasticity of system application is also not satisfactory for most users. Therefore in practice almost all users of artificial neural network prefer software solutions accepting their limitations.

Looking on Fig. 1.28 we can see, that some of biological neural systems can be located within artificial neural networks range. For example “brain” of the shrimp can be compared with artificial neural network and have no superiority nor in the complexity domain, nor in sense of speed of information processing. Therefore red dot symbolizing shrimps neural system parameters is located

inside yellow area symbolizing artificial neural networks parameters. But for most biological species complexity of their neural systems and speed of data processing are much greater than best parameters achieved by artificial neural networks. In case of human brain its complexity is **billion** times greater, than parameters observed in artificial neural networks. Therefore thinking about natural and artificial neural networks we should be very humbly!

1.9. What are main advantages of neural networks, who uses them and what are they used for?

(Translation by Ryszard Tadeusiewicz; rtad@agh.edu.pl)

TRANSLATION WILL BE ADDED SOON!

1.10. Is neural networks going to displace traditional computers?

(Translation by Daniel Bohusz daniel.bohusz@gmail.com)

As one could find out from the specification given in the last subsection - there are many good ways of using the neural networks. But does it really mean that one should give up "classical," well known computers and solve all the computational problems by the use of the network?

Despite the fact that the network is fashionable and useful device it has lots of important limitations. All those limitations are going to be distinguished in the following chapters in which one can find out about some other structures of the network, methods of learning it and ways of using it with specific tasks. Still, it is possible to provide a few characteristic features of the tasks, which cannot be solved by the use of the neural network.

The first category of such tasks, with which one should give up the neural network, are tasks connected with symbol manipulation. Any form of information processing in the form of symbols is extremely difficult to deal with by the use of the neural network, therefore if there is the element based on symbol processing such element should be a sign that one should not use the neural network. To sum up it is possible to stress that it does not make any sense to create text editor or algebraic expression processor which would work then based on the neural network.

The next "classical" areas, in which one should resign from the neural network are problems connected with the issue of calculations that require high precision of numerical results. The network always works in qualitative way, which means that the results given by the network are always approximate. The precision of such approximation is quite satisfactory in many applications, like for example: signal processing, picture analysis, speech recognition, prognosis the value of quotation, robots controlling, approximation of the value of compound function etc. However, it is absolutely impossible to do approximate calculations characteristic to the bank account service or in the case of precise engineering calculations.

The last case, in which one should not expect proper results when using the neural network, is connected with tasks that require many stages of reasoning like, for example the stages of deciding about authenticity or falseness of some sequences of logical statements. The network usually solves

the problem using one way - if it makes to so solve the problem, the result is shown immediately and becomes a great practical success. If, on the other hand, it is necessary to carry out a kind of argumentation and what is more if it is necessary to provide the documentations of the conclusions (for example expert system) - the network becomes useless and any attempts of using it leads to frustrating failures.

1.11. So maybe it's not worth to occupy oneself with the neural network?

(Translation by Daniel Bohusz daniel.bohusz@gmail.com)

From the above-mentioned statement one should not jump to conclusions: as a matter of fact, the network cannot make symbolic calculations by itself, but it can support the systems which operate on symbols on the basis of functions with which the systems cannot deal by themselves. A good example of such networks are Teuvo Kohonena networks or the classical network NetTalk Terence Sejnowski that is used to change the orthographic text into the phonematic symbol sequence that is served to controlling speech synthesizers. Similarly, from the statement that the net is useless when used with the terminals in banks one should not conclude that the net is completely useless to banks themselves, as the net turned out to be exceptionally useful when dealing with credibility of debtors or establishing conditions of negotiating contracts. There are much more similar reservations but the most important is the final conclusion: The neural network is useful with many applications, it is, however, not as universal as the classical computer.

Therefore, the enthusiasts can easily indicate the tasks, which when solved by the use of the neural network turned out to be much better that those solved by the use of classical computers. Malcontents, on the other hand, can easily indicate that the tasks solved by the use of the network turned out to be incorrect. Both what the enthusiasts and malcontents say can be truth, and therefore we shall try to find the truth in the following chapters by describing the technique of the neural network, ways of learning it and also by describing simple programs, which are attached to the book and which I prepared as to enable everyone to check how much this all is worth.

In this chapter I explained the genesis and good points of the neural networks. Both the genesis and good points can turn out to be the reason for you to know something more about that device. There are people to whom a favorable stock market prognosis is more important than any other advantage of the network, and maybe you are just like them. Let me stay however by my fascinations and, when writing about the neural networks that are treated as a device used to more or less useful practical goals, I shall show you how interesting they are, as they are based on the structure and functioning of the live brain. I shall describe other significant success achieved by the engineers, economists, geologists and doctors using the network to improve their tasks, I shall show you how useful the network was to them when they were dealing with problems. I shall also pay your attention to consequences of the studies leading to understanding our own mind. You will find out how many conclusions you can make when dealing with the network working as a self educating system. By the access to my programs on the Internet (the address is given in the preface) you will be not only a witness but also an active participant of the surprising experiments which will allow you to know the neural network much better and by that to know even better your own brain; mysterious and

abounding in various capabilities, and about which Shakespeare wrote that it is …*a fragile house of soul* ….

Then: do you know now why it is worth to occupy yourself with the neural networks?

If yes and if you still want to know them this book will help you to fulfill your purpose. I invite you to reading further chapters! The text of this book is current and exhaustive, which means that I did my best to present in it all those what is the most important and the most recent in the neural networks. It is, however, deprived (on purpose) of those numerous, small but onerous details in which each branch of calculation technology abounds and which details can be usually expressed by the use of mathematical formulae. I decided to treat seriously the principle written on the cover of the book by Stephen W. Hawking "*The Short History of Time*" which says that each equation included in the text diminish half the number of readers. It is important to me that this book is read by many people, among which there surely are those who when reading it may find the neural networks fantastic and useful device. Therefore, I am not going to include in the main text of this book not even one equation, because if there was at least one reader who could become discouraged by that I would consider that heavy loss. In each chapter, on the other hand, the most important theoretical information are given in separate frames, and therefore you can use them if you want to necessarily know the less colorful but more important, highly mathematical technique of the neural networks.

1.12. Control questions and self-work tasks

(Translation by Daniel Bohusz daniel.bohusz@gmail.com)

1. Enumerate a few exemplary problematic areas, except for the neural networks, which are fashionable and willingly used in contemporary informatics.

2. Which functions of the brain were known as first and how was that knowledge acquired?

3. Who was the first to prove that the brain is made of enormous, combined networks, and, on the other hand, anatomically and physiologically separate neurons?

4. What animal was used to acquire the basic information concerning electrochemical functions of the brain, thanks to which Hodgkin and Huxley worked out the detailed model of those functions and were awarded the Prize of Nobel?

5. Enumerate the basic parts of natural (biological) neuron.

6. Why are synapses so important in the description of the functioning of the neural networks (biological and natural)?

7. What are neurotransmitters? Can you provide some of their names and characteristics of those useful compounds?

8. Why the artificial neural networks are usually made of layers?

9. How are the neurons of individual layers of the networks combined and why are they combined that way?

10. What are the modern methods of researches that are used with acquiring the knowledge about the structure and functions of the brain and other biological neural structures?

11. Enumerate a few exemplary areas with the use of the neural networks.

12. What are the advantages of the neural networks over the typical computers and what are their disadvantages?

2. A neural net structure

2.1. How is it build?

(Translation by Agata Barabasz agata.barabasz@op.pl)

Who of us have not started to get to know the world by taking the alarm clock into pieces or crushing the tape recorder – just to find out what was inside?

However, before I tell you how a network works and how to use it – I will try to a certain extent precisely and simply describe it to You how it is built.

As you already know from the previous chapter – a neural network is a system, which makes specific calculations, based on simultaneous activities of many connected with each other elements called neurons.

Such a structure has been at first observed in a biological nervous system (for example in human's cerebellum, a part of which I have schematically presented in Fig. 2.1).

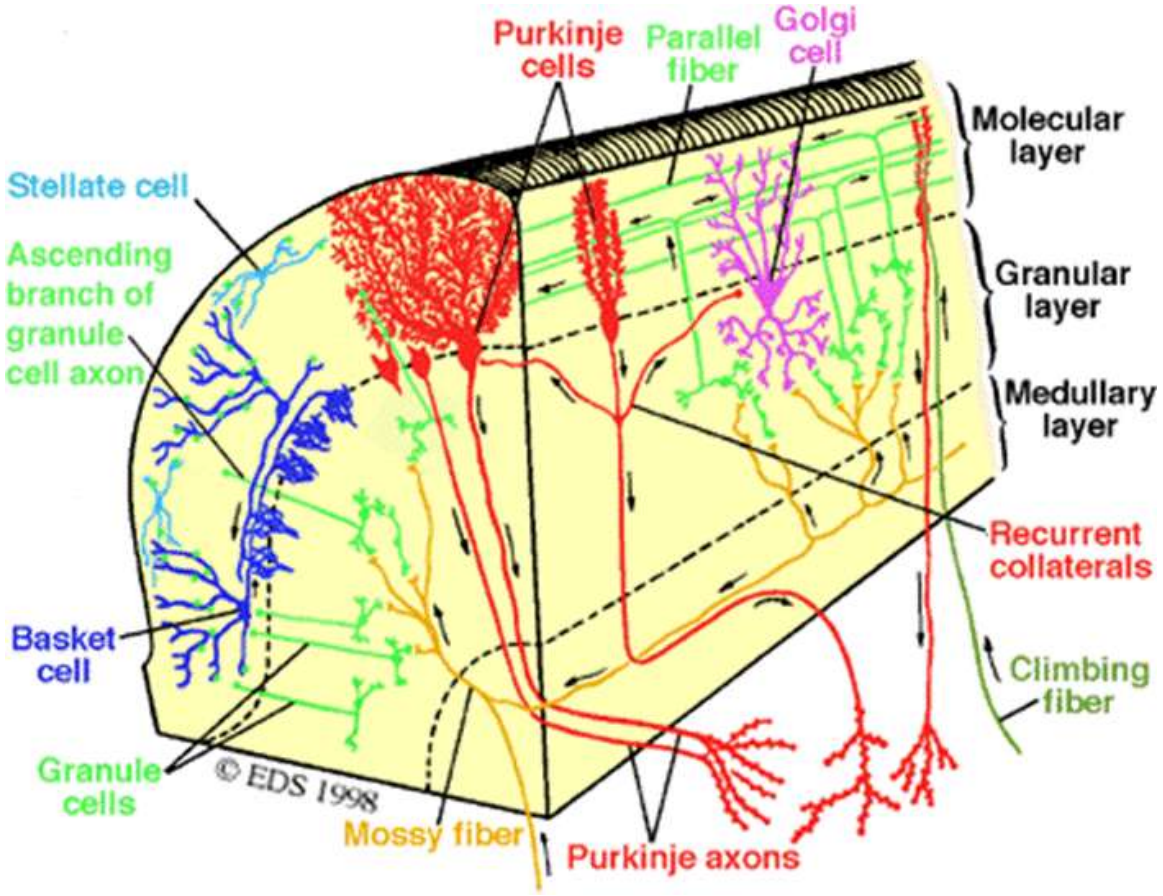


Fig. 2.1. Scheme of cerebellar cortex shows, that biological nervous system is build of many connected neurons. The same pattern is applied in case of artificial neural nets.

Neural networks are also built from a lot of neurons, but these are artificial, i.e. much more simplified than the original ones, and also connected in a less complicated (more primitive) way. - The artificial neural network model of the real nervous system structure would seem to be rather unclear and difficult to be controlled. Fig. 2.2 shows how it would look like the artificial neural net based on the identical structure schemes as those appearing in the real nervous system.

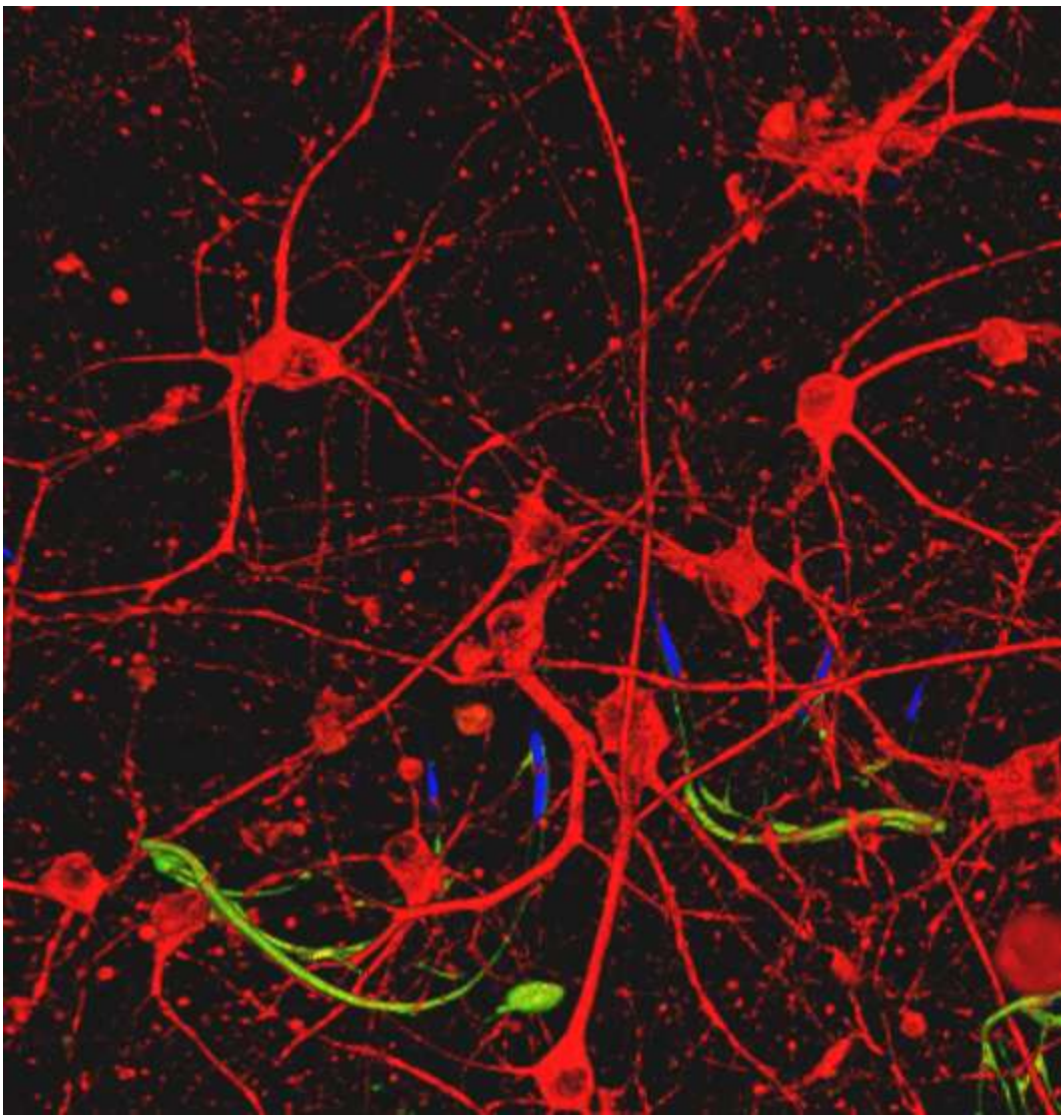


Fig. 2.2. The artificial neural net with structure founded on the 3D brain map would not be comfortable to use in practice. (source: http://www.kurzweilai.net/images/3D-nanoelectronic-neural-tissue.jpg)

As you notice such a structure does not seem to be friendly to experiment with. On the contrary - it is easy to be lost like in a forest.

So artificial neural nets we built in such a way that their structure could be easy traced and cheap in an implementation. In a result they should be flat (not three-dimensional) and should have given are a regular structure, with layers of neurons which all have well defined objectives and are linked according to simple, however a wasteful rule of connecting “everyone with everyone” (see Fig. 2.3).

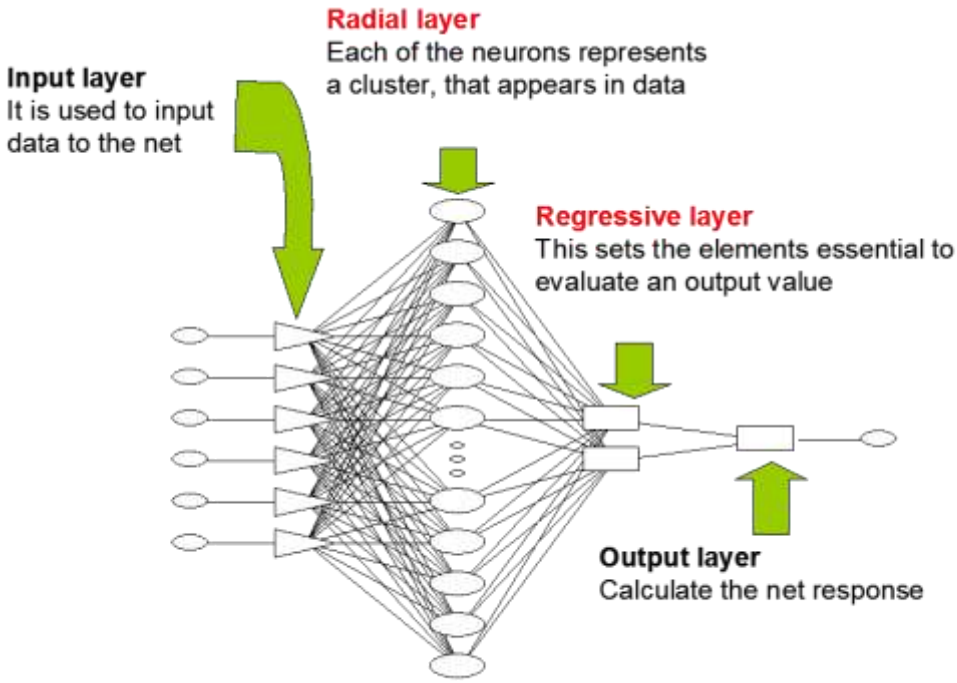


Fig. 2.3. Scheme of practically used neural net (GRNN type) shows, that it's structure is strongly rationalized and simplified in comparison to the biological one.

The following three factors decide about neural networks properties and possibilities:

a) from which elements a network is built (i.e. how artificial neurons look like and how they work);

b) how these elements are connected with each other;

c) the way of establishing the parameters of a network by its learning process.

We will be successively consider these factors.

2.2. How to make an artificial neuron?

(Translation by Agata Barabasz agata.barabasz@op.pl)

The basic "building materials" that we use to create a neural network are artificial neurons.

Now we will try to learn about them more precisely. In the previous chapter you have seen some pictures illustrating the shape of a biological neuron, but it will not harm to recall one more picture, so see in Fig. 2.4, how an exemplary neuron (simplified) looks like.

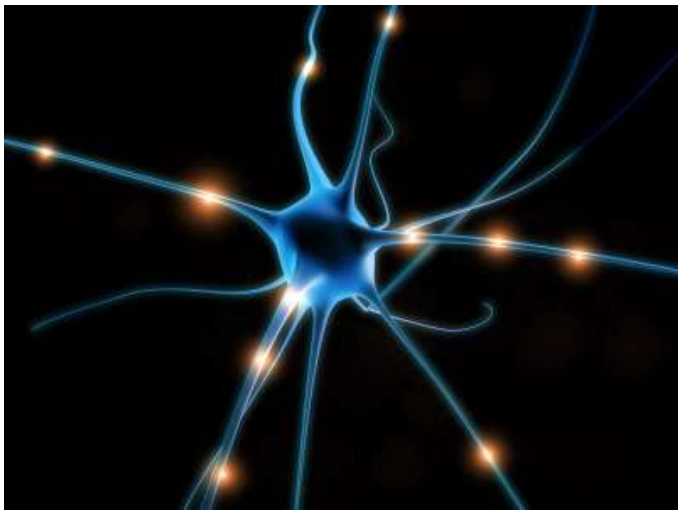


Fig. 2.4. Structure of a biological nerve cell (a neuron). (Source: http://cdn.thetechjournal.com/wp-content/uploads/HLIC/8905aee6a649af86842510c9cb0fc5bd.jpg)

So that you do not think that all real neurons look exactly like that, in picture 2.5 I am showing you one more illustration of a real biological neuron, dissected free of a rat's cerebral cortex.

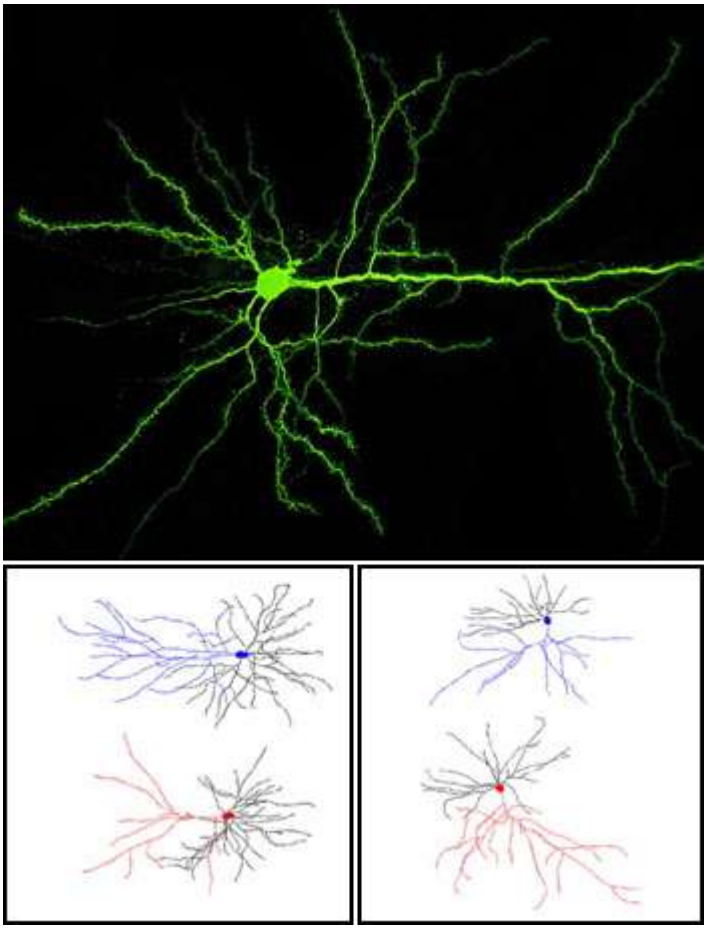


Fig. 2.5 Microscopic views of real neurons (source: http://newswire.rockefeller.edu/wp-content/uploads/2011/12/110206mcewen.1162500780.jpg)

It is hard in this picture to guess, which of the many visible on it fibers is an **axon**, which is always single and as the only one delivers signals **from** the given neuron to all the others, and which are performing the role of **dendrites**.

Nevertheless, this is also a real biological neuron, thus also such a cell as this one our artificial neuron has to map well, and of which we will take care now more precisely.

Artificial building neurons used in the networks technique are of course very simplified models of nerve cells, that occur in nature.

A structure of an artificial neuron best illustrates the scheme presented in Fig. 2.6. Comparing this illustration with figures 2.4 or 2.5 you will realize, how far neural networks' researchers simplify biological reality.

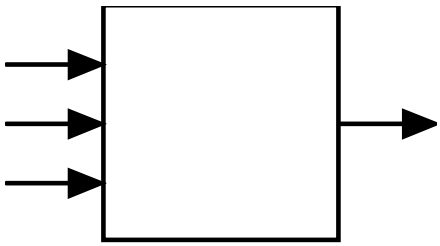


Fig. 2.6 General scheme of an artificial neuron shows the degree of its simplification

However, in spite of this simplifications artificial neurons keep all these features, which are valid from the point of view of tasks we want to entrust them within built networks, being the computer science tools, rather than biology models.

- Firstly, **they are characterized by having many inputs and one output**. The input signals x_i (i = 1,2,…,n) and the output signal y may take on only numerical values, generally of the range from 0 to 1 (sometimes also from –1 to + 1), whereas the fact that within the tasks being solved by networks they represent some information (e.g. as the output of a decision, who has been recognized by the neural network, which has been analyzing someone's photo), is the result of a specific **agreement**. Generally particular meanings are ascribed to network's input and output signals in such a way that the most crucial is this, on which input or output a given signal has occurred (each input and output is associated with a specific **meaning** of a signal), additionally signals scaling is used, so selected that signal values that would be circulating in a network, would not be out of an agreed range – e.g. from 0 to 1.

- Secondly – artificial neurons perform specific activities on signals, which they receive on inputs, as a consequence they produce signals (only one by each single neuron), which are present on their outputs and are sent forward (to other neurons, or onto this network's output, as the solution of a raised problem). Network's assignment, reduced to the functioning of its basic element, which is a neuron, is based on this that it transforms an input data x_i into a result y applying rules resulting from that how it has been built, and what has been taught. Considered up to this point neuron's properties have been illustrated on figure 2.7.

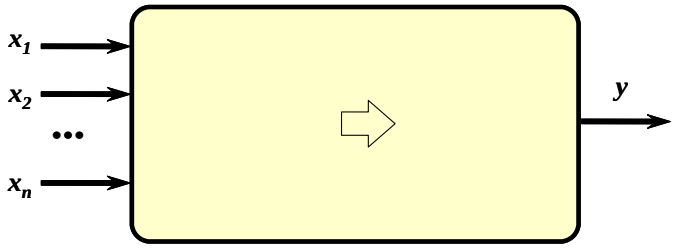


Fig. 2.7 Basic signals occurring in a neuron

- Thirdly – neurons may learn. This purpose serve w_i coefficients called **synaptic weights**. As you certainly remember from the previous chapter – these reflect rather complicated biochemical and bioelectric processes, which take place in real biological neuron's synapses. From further considerations point of view the most significant is that synaptic weights can be modified (i.e. their values can be changed),

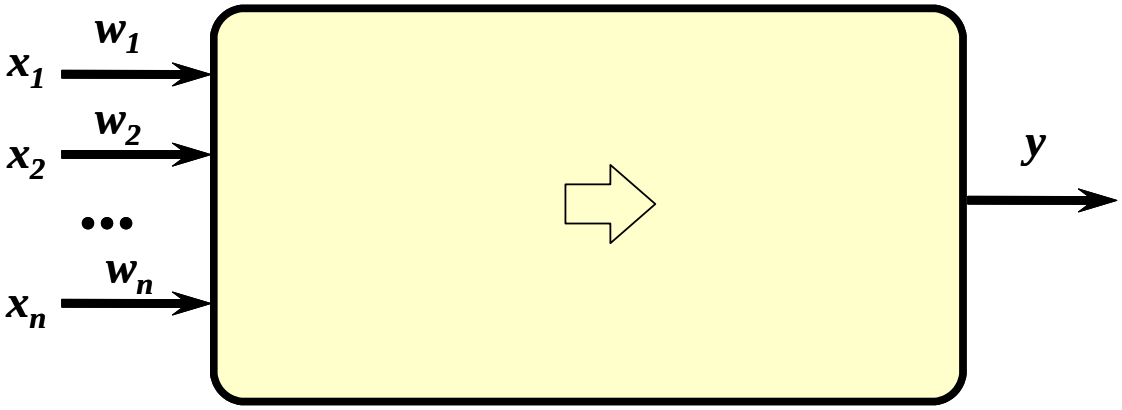


Fig. 2.8 Adding to a neuron's structure adjustable weight's coefficients makes it a learnable unit

what constitutes a basis for teaching networks. A scheme of a neuron capable of learning has been shown on figure 2.8.

Summing up this discourse it could be ascertained that artificial neurons can be treated as elementary **processors** with the following features:

- each neuron receives **many input signals** x_i and on their basis determines its own "answer" y, that is produces **one output signal**;
- with each separated neuron's input is connected a parameter called **weight** w_i. This name means that it expresses a degree of significance of an information arriving to this neuron through just this input;
- **a signal**, coming in through a particular input is first **modified** with the use of the weight of that given input. Most often a modification is based on this that a signal is simply **multiplied**

through the weight of a given input, so in consequence in further calculations it is already participating in the modified form: **strengthened** (if the weight is greater than 1) or **restrained** (if the weight's value is less than 1). A signal from a particular input may occur even in the form **opposite** in relation to signals from the other inputs, if it's weight has a negative value. Inputs with negative weights are among neural networks users defined as so called **inhibitory inputs**, whereas these with positive weights are called **excitatory inputs**.

- input signals (modified by adequate weights) are **aggregated** in a neuron (see figure 2.9). Once again considering networks in general, many ways of input signals aggregation may be given, nevertheless most often it is based on this that signals are simply **summed up** giving as the result some helpful internal signal, called a cumulative neuron stimulation or a postsynaptic stimulation. This signal may be also defined as a net value.

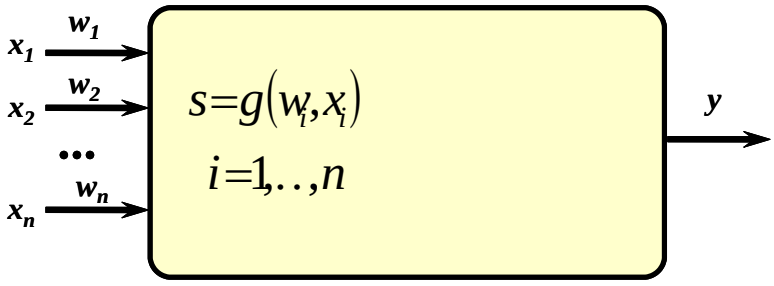


Fig. 2.9. An aggregation of input data as the first of neuron's internal functions

- to so created sum of signals the neuron adds sometimes (not in all networks' types, but generally often) some extra component independent of input signals, called a **BIAS**.

A bias, if it is taken into account, also undergoes a learning process, that is why sometimes one can imagine, that a BIAS is an extra synaptic weight associated with the input, on which it is provided an internal signal of constant value equal to 1. A BIAS role lies in this that thanks to its presence during a learning process a neuron's properties may be formed in a much more free way (without having it the aggregation function characteristics always must pass through the beginning of the coordinate system, what sometimes is a burdensome " anchor"). A scheme of a neuron, in which a BIAS has been taken into account, is shown in figure 2.10;

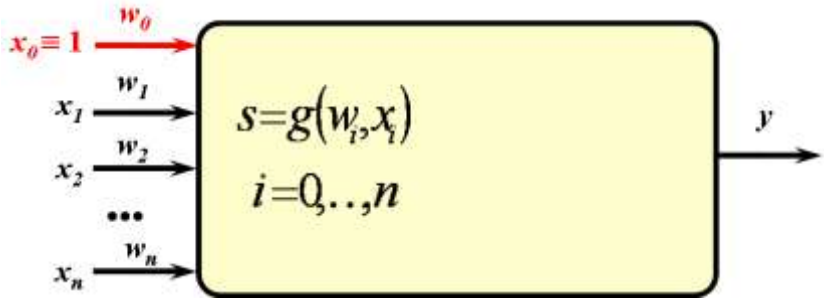


Fig. 2.10. The application of the additional parameter, which is BIAS

- A sum of internal signals multiplied by weights plus (possibly) a bias may be sometimes sent directly to its axon and treated as a neuron's **output signal**. In many types of networks that is enough. In this way work so called linear networks (for example a net named **ADALINE** = ADaptive LINEar). However, in networks with richer abilities (for example in very popular networks called **MLP** from the words Multi–Layer perceptron) a neuron's output signal is calculated by means of some nonlinear function. This function in the whole book we will be designating with the symbol *f( )* or *φ ( )*. A scheme of a neuron including both an input signals' aggregation and an output signal's generation is presented in figure 2.11;

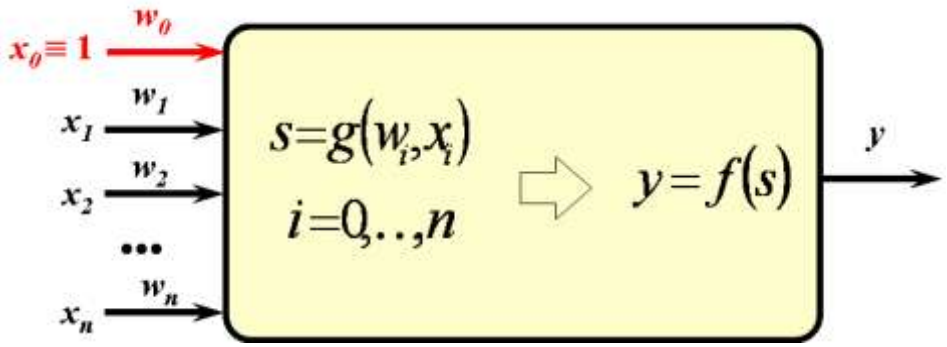


Fig. 2.11. The full complete of neuron's internal functions

- a function *φ ( )* is called a **characteristic of a neuron** (a transfer function). There are known many different neuron's characteristics, what illustrates figure 2.12 Some of them are chosen in a such way that artificial neuron's behavior would be the most similar to a real biological neuron's behavior (a sigmoid function), but they also could be selected in such manner, which would assure the maximum efficiency of computations carried on by a neural network (a Gauss function). In all the cases function *φ* ( ) constitutes an important element going between a joint stimulation of a neuron and its output signal;

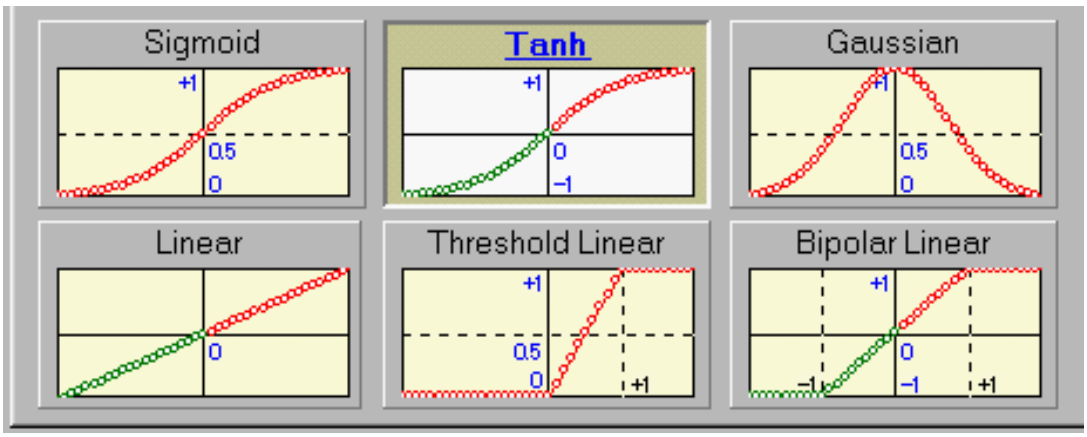


Fig. 2.12. Some of the more often used neuron's characteristics

- a knowledge of the input signals, weights' coefficients, inputs aggregation method and neuron's characteristic, allow to unequivocally define at any time it's output signal, with usual assuming that (in contrast to what takes place in real neurons) this process occurs **immediately**. Thanks to this in artificial neural networks changes of input signals are practically immediately appearing on output. Of course this is a clearly theoretical assumption, because after input signals change even in electronic realization some time would be needed for establishing the right value of an output signal by an adequate integrated circuit.. . Much more time would be necessary to achieve the same effect in a net working as a simulation model, because a computer imitating network activities must then calculate all values of all signals on all neurons outputs of this network, what even on very fast computers could take a lot of time. While speaking about a prompt neuron's action I mean that considering network's functioning we will not pay attention to a factor, which is a time of neuron's reaction, because this will be insignificant for us. A complete structure of a single neuron is presented in figure 2.13.

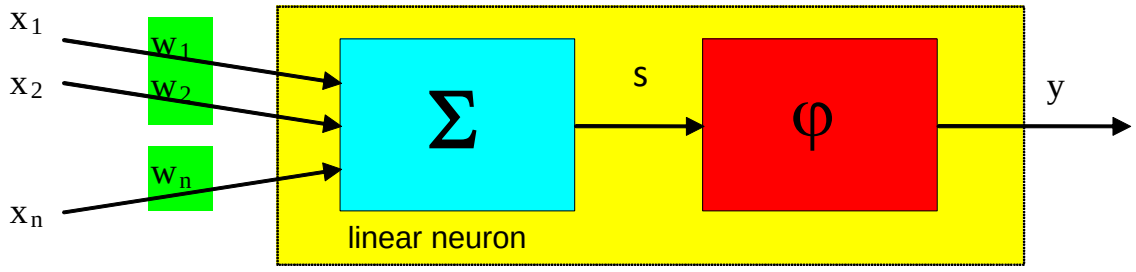


Fig. 2.13. Structure of a neuron as a processor, which is the basis for building neural networks

A neuron presented in this picture is the most typical "material", which is used for creating a network. More precisely – such typical "material" is a neuron of a network defined as MLP (Multi–Layer Perceptron), the most crucial elements of which I have collected and presented in figure 2.14. It is visible in this picture that neuron MLP is characterized by the aggregation function consisting of simple summing up the input signals multiplied by weights, and uses a nonlinear transfer function with a distinctive sigmoid shape.

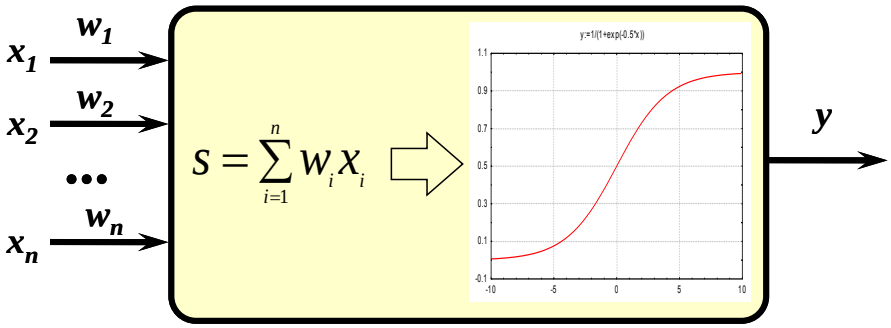
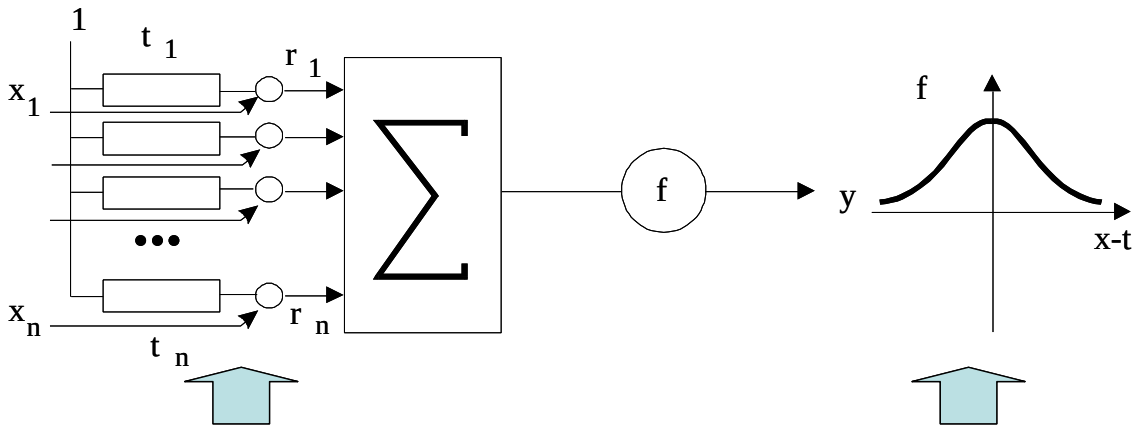


Fig. 2.14. The most popular component of neural networks – the MLP type neuron

However sometimes in neural networks for special purposes there are used so called **radial neurons**. They have an atypical method of input data aggregation, and they also use an untypical characteristic (a Gauss's one) and are taught in an unusual way. At this moment I do not intend to elaborate on a subject of this specific neurons, which are used mainly to create special networks called RBF (Radial Basis Functions), but in figure 2.15 I present a scheme of such a radial neuron, to enable you to make a comparison with discussed earlier a typical neuron shown in figure 2.14.



In this type of neuron the aggregation of the input signals consists of evaluating **the distance** between the present input vector **X** and the **centroid** of a certain **subset T** determined during a teaching process

Also a nonlinear transfer function in these neurons has a different form – "bell-shaped" **gaussoid** - i.e. it is a **non-monotonic** function.

Fig. 2.15. A structure and peculiar properties of a radial neuron, denoted also as RBF

2.3. Why do not we use an exact model of a biological neuron?

(Translation by Agata Barabasz agata.barabasz@op.pl)

All artificial neurons, sigmoid and radial, described in this chapter, as well as used in further parts of this book, are **simplified** models of real biological neurons. This statement has already appeared, however now I want to show you, how far simplified artificial neurons are. To achieve this purpose I will use the example of researches conducted by de Schutter. For many years this researcher dealt with that, to maximally faithfully and maximally exactly reconstruct in the computer model, all that we know about the structure and working (in the smallest details) of the only **one** neuron - specifically so called Purkinji cell. His model referred to electric systems, which according to Hodgkin's and Huxley's (the Nobel Prize in 1963) researches model a bioelectrical activity of individual fibers (dendrites and axon) and a cell membrane of neuron's soma. In the researches with the extraordinary accuracy a shape of the real Purkinji cell was reconstructed and Neher's & Sakamann's researches (the Nobel Prize in 1991) about functioning of so called ion channels were

taken into consideration. A structure of the modeled cell and replacement circuit diagram, used in de Schutter model are shown in a figure 2.16.

Fig. 2.16 The Model of the neuron maximally faithful to a biological original, used in de Schutter's researches (source: http://homepages.feis.herts.ac.uk/~comqvs/images/purkinje_padraig.png)

The model built by de Schutter, turned out to be extremely complicated and costly in calculations. It is enough to say, that for building the model were used:

- ⇒ 1600 so-called **compartments** (fragments of the cell treated as homogeneous parts containing determined substances in determined concentrations),
- ⇒ 8021 models of ion channels,
- ⇒ 10 types of different complicated mathematical descriptions of ion channels dependent on voltage,

⇒ **32000 differential equations (!)**
⇒ **19200 parameters** necessary to estimate at tuning the model up,
⇒ a precise description of the morphology of a cell reconstructed on the basis of precise microscopic images

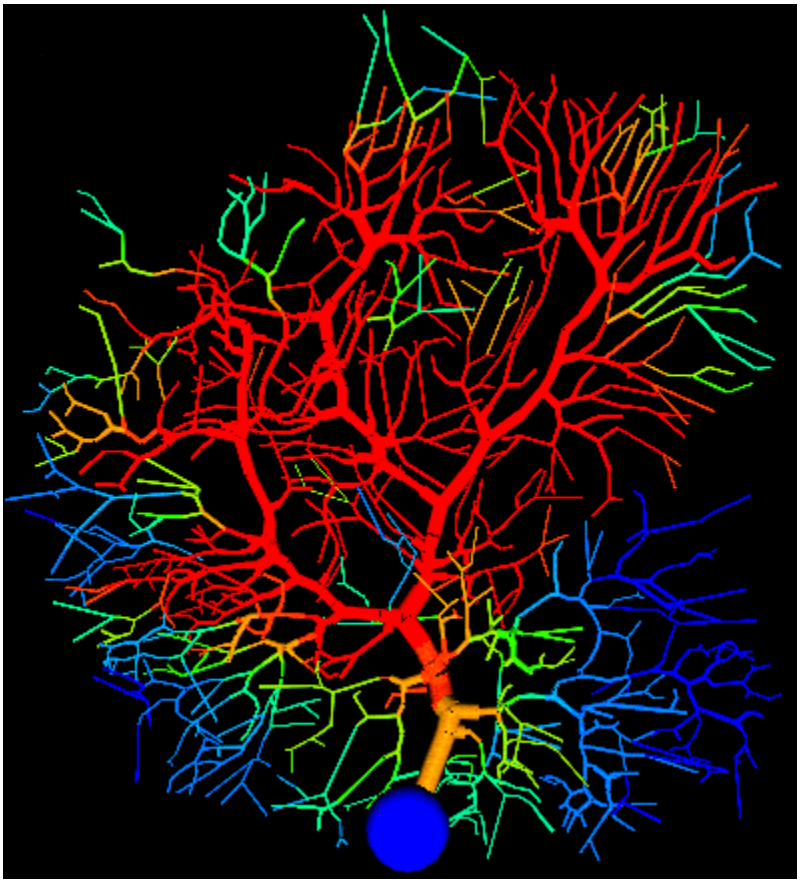


Fig. 2.17. Example result obtained in de Schutter researches.
(http://www.tnb.ua.ac.be/models/images/purkinje.gif)

Nothing strange, that for simulating several seconds of "life" of such a nerve cell was necessary to use a large supercomputer , yet it required many hours of his continuous work.

It has to be admitted, that results of this modeling are very impressive. One of them is presented in figure 2.17.

However results of this experiments are unambiguous: The attempt of faithful modeling the structure and action of a real biological neuron turned out to be successful but it is too expensive way in order to try to create **practically useful** neural networks this way. Therefore, from now on we will be using only simplified models, and we will be expecting, that in spite of these applied simplifications the neural network will be able to not only effectively solve different tasks, but additionally it will also be able to the fact, that it's behavior can bring us interesting conclusions about the behavior of human (for example your!) brain. Soon you will convince yourself about it!

2.4. How does an artificial neural network work?

(Translation by Leszek Pstras lpstras@hotmail.com)

From the earlier description of neural networks it follows that each neuron possesses certain internal memory (represented by the values of current weights and bias) as well as certain abilities of converting input signals into the output signal. Although these abilities are rather limited (this is why a neuron is a relatively low cost processor and one can build a system consisting of hundreds or thousands of such elements), they turn out to be sufficient for building systems performing very complex data processing tasks.

As a result of a limited amount of information gathered by a single neuron (having usually not many adjustable weights) as well as of very poor computing capabilities of a single neuron (only aggregating signals in order to calculate the output signal), a neural network usually needs to consist of several neurons and can act only as a whole. Therefore, all capabilities and properties of neural networks discussed earlier result from collective performance of many connected elements (the whole network as opposed to a single neuron) what justifies the name used sometimes for this part of computer science – **MPP** (*Massive Parallel Processing*).

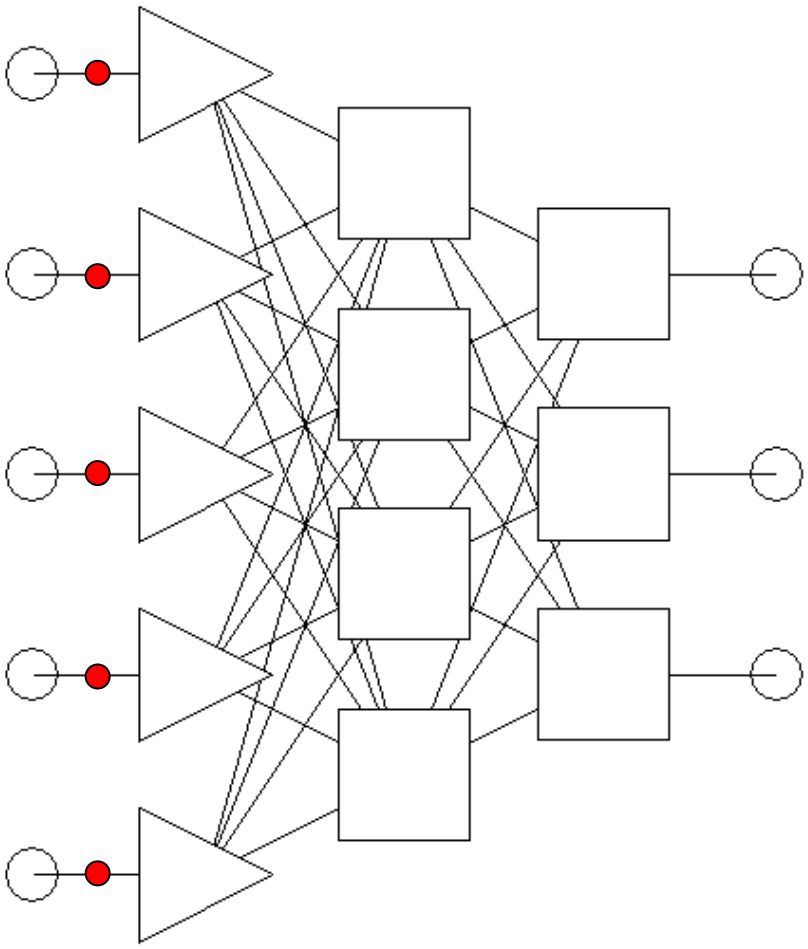


Fig. 2.18. The neural network starts working when the signals carrying a new task appear at the network input ports (red dots).

Let us now look in more details into operation of the whole neural network. As one can see from the above remarks, both the network program and the information constituting the knowledge database, as well as the data being calculated and the calculation process itself are all completely distributed. It is not possible to point the area in which such and such information is stored, even though neural networks are being used as memory, especially as so called associative memory and as such perform very well. It is also impossible to relate certain area of the network to the selected part of the algorithm being used, for instance to indicate which network elements are responsible for initial processing and analysis of input data and which elements produce final network results.

We will now analyze how a neural network works and what roles do the single elements play in the whole network operation. In our analyses we assume that all network weights are already determined, which means that the teaching process has been accomplished. Teaching the network, which is very important but also relatively difficult to understand, will be dealt with in the following chapters. We will start the analysis from the point where a new task is presented to the network. The task is represented by a number of input signals appearing at all network input ports. In fig. 2.18 these signals are represented by red dots.

The input signals reach the neurons in the input layer, which usually do not process the signals but only distribute them, in other words - send them to the neurons in the hidden layer (fig. 2.19). By the way, it is worth mentioning that the distinct nature of the neurons in the input layer, which only distribute the signals rather than process them, is very often presented on the neural network schemes by different type of graphic symbols representing those neurons (for instance a triangle instead of a square, as shown in the figures here).

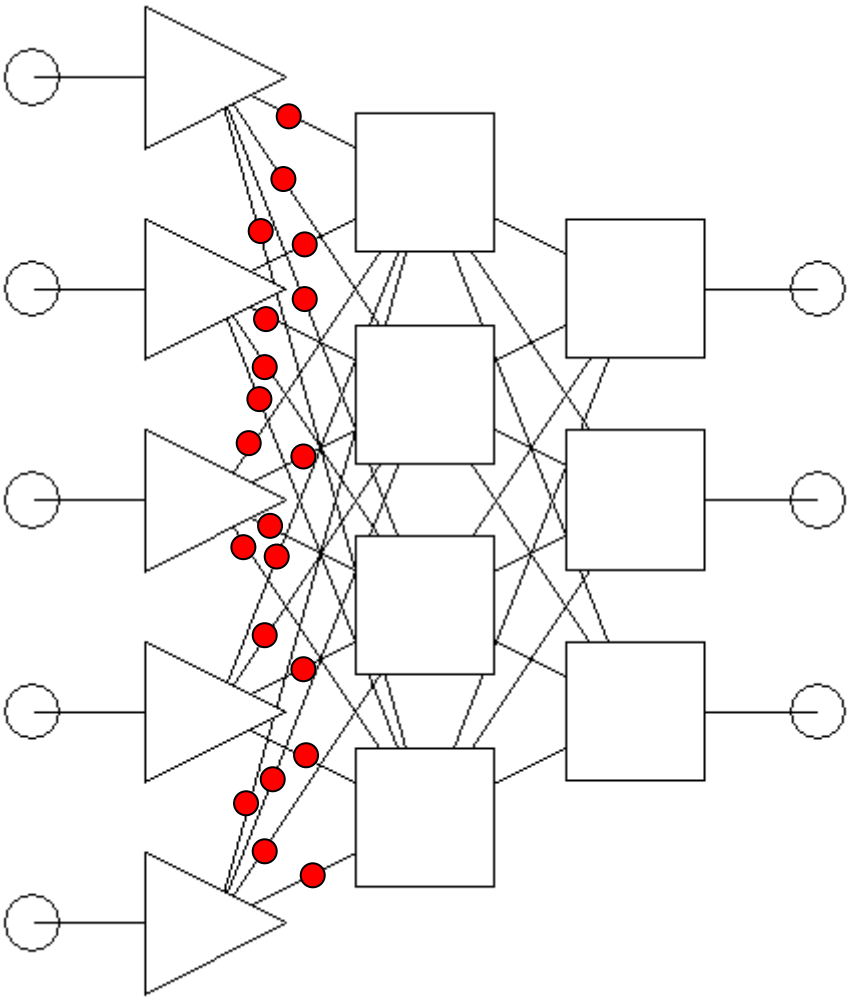


Fig. 2.19. The input signals (unprocessed by the input layer) are sent out to all neurons in the hidden layer

Next stage involves activation of the neurons in the hidden layer. These neurons, using their weights (hence utilizing the data stored in them), firstly modify the input signals, secondly aggregate them and then, accordingly to their characteristics (in fig. 2.20 shown as sigmoid functions), calculate the output signals which are subsequently directed to the neurons in the output layer. This stage of data processing is crucial for neural networks. Even though the hidden layer is 'invisible' outside the network (its signals cannot be registered neither at the input ports nor at the output ports of the network – hence the name), it is there where most of the task solving is being performed. The most of the network connections and their weights are located just between the input and the hidden layers, thus one can say that the most of the data gathered in the teaching process is located exactly in this layer. The signals produced by the hidden layer neurons do not have any direct interpretation unlike input or output signals of which every single one has a meaning in terms of the task being solved. However, using the manufacturing process analogy, one can say that the hidden layer neurons produce semi-products, that is signals characterizing the task in such a way that it is relatively easy to use them afterwards in order to 'assemble the final product', that is to find the final solution by the output layer neurons (fig 2.20).

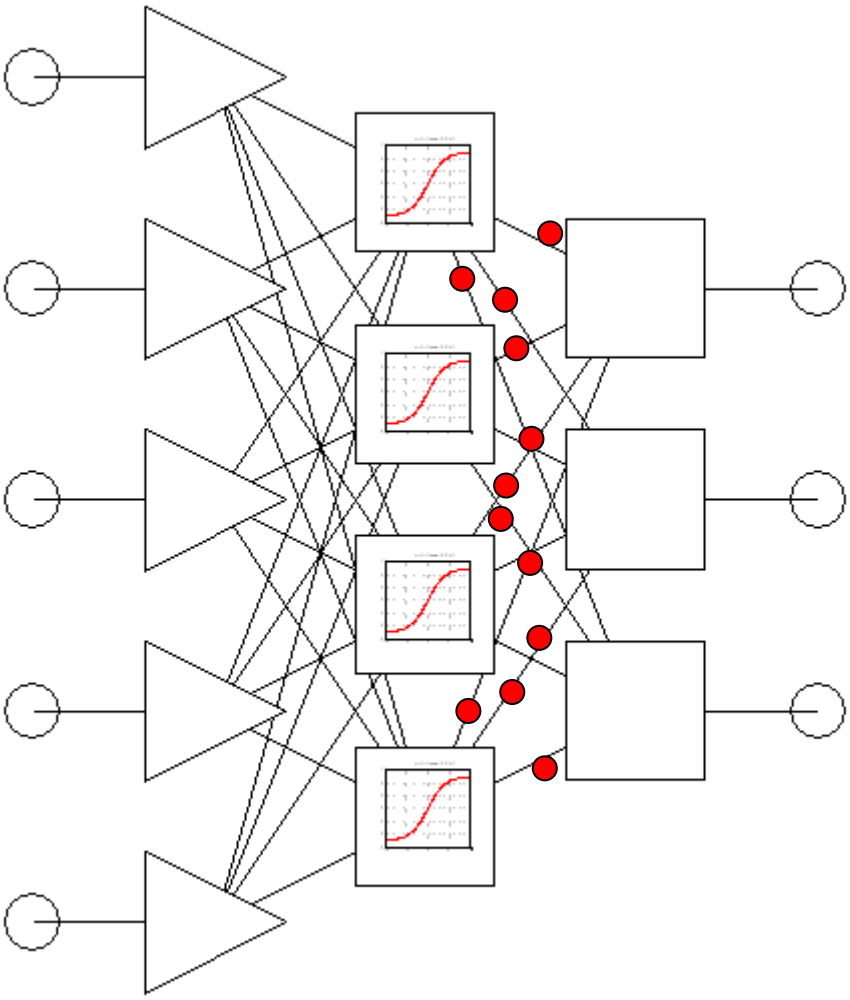


Fig. 2.20. After processing the signals, the neurons from the hidden layer produce intermediate signals and direct them to the neurons in the output layer

Following more precisely the performance of the network at the final stage of task solving, it can be noticed that the output layer neurons take advantage of their abilities to aggregate signals as well as of their characteristics in order to build the final solution given at the network output ports (fig 2.21).

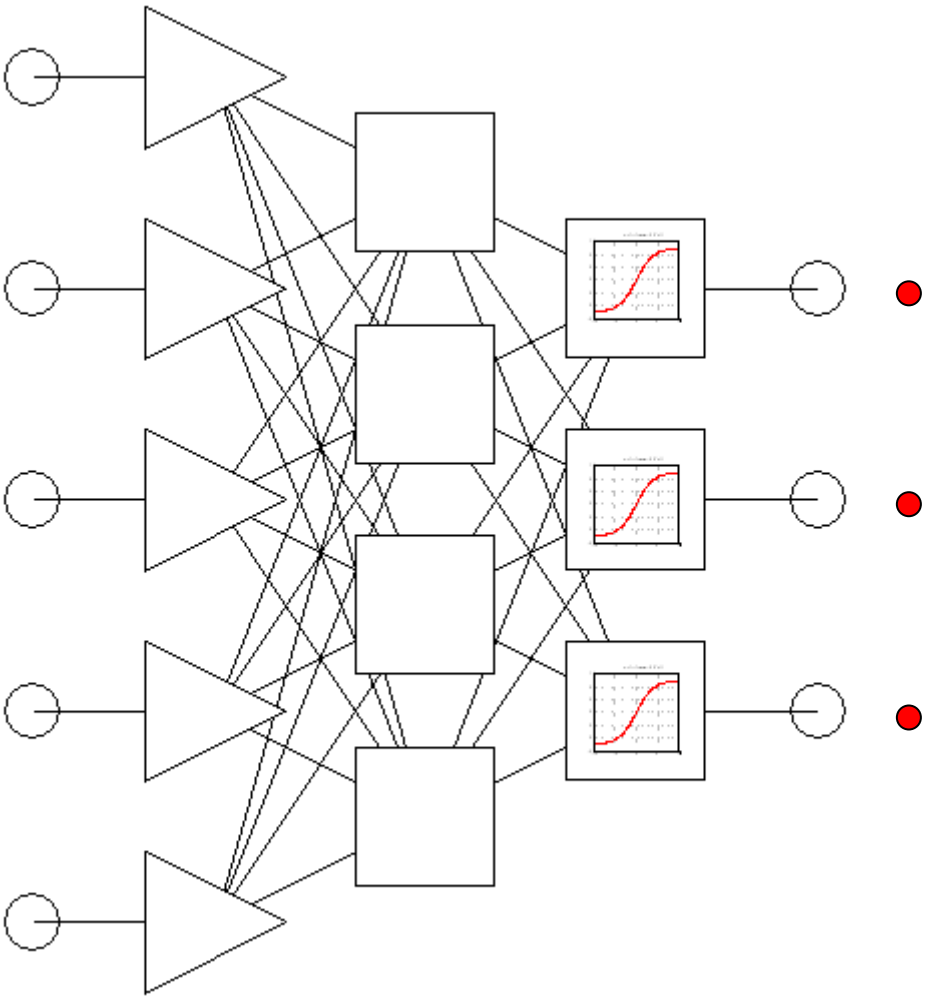


Fig. 2.21. The neurons from the output layer use the information given by the neurons from the hidden layer and calculate the final results of the task being solved

To repeat: the network always works as a whole and all its elements contribute to performing all the tasks within the network – just like it is done with the hologram reproduction where from each piece of a broken photographic plate one can reproduce the whole picture of the photographed (hologrammed?) object.

One of the consequences of such a network performance is its unbelievable ability to work properly even after a failure of a significant part of its elements. There was a neural network researcher (*Frank Rosenblatt*) who used to teach his own networks certain abilities (for instance letters recognition) and then test them while damaging more and more of its elements (the networks were realized as special electronic circuits). It turned out that he could damage a significant part of the network and it would still perform properly (fig 2.22). Failure of a higher number of neurons and connections would though deteriorate the quality of the network performance but only in that the damaged part of the network would make more mistakes (say, it would recognize the letter O as D) – but it would not refuse to work.

Compare this behavior to the known fact that in most electronic equipment (computers, TVs) a failure of only one element can cause the system to stop working, while in human's brain thousands of neurons die every day (from different reasons) and still our brains work as a whole unfailingly for many years.

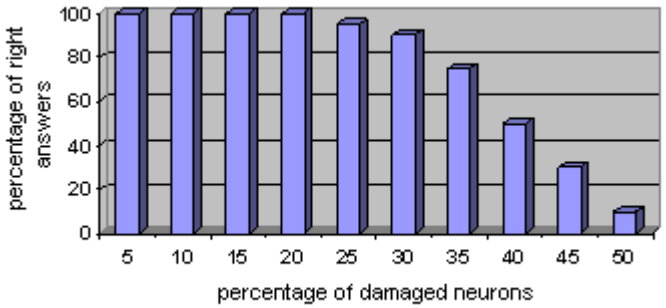


Fig. 2.22. Neural networks have an astonishing feature: they can work properly even if some of its elements are damaged!

Details about this fascinating and rarely discovered property of neural networks are described in paper: Tadeusiewicz R., Figura I.: *Phenomenon of Tolerance to Damage in Artificial Neural Networks*, Computer Methods in Material Science, vol. 11, Nr. 4, 2011, pp. 501-513

2.5. How does neural network structure affect its capabilities?

(Translation by Łukasz Brodziak; lukasz.brodziak@gmail.com)

Let's consider now the relationship between the structure of a neural network and the tasks which it is able to perform. As you already know neurons described in previous chapters are used to create the network. The structure of such network is created by connecting outputs of neurons with inputs of the following ones(according to chosen scheme) this creates a system capable of parallel and fully concurrent processing of various information. For previously mentioned reasons we usually choose layer-structured network and connections between the layers are made according to „each to each" rule. Obviously specific topology of the network, meaning mainly the amount of chosen neurons in individual layers, should be derived from kind of the task we want the network to perform. In theory the rule is quite simple: the more complicated is the task, the more neurons in the network are needed to solve it, as the network with more neurons is simply more intelligent. In practice, however, it is not as unequivocal as it would seem.

In vast literature considering neural network one can find numerous works which prove that, as a matter of fact, the decisions regarding network's structure affect its behavior considerably weaker than expected. This paradoxical statement derives from the fact that behavior of the network is determined in fundamental way by the process of network teaching and not by the structure or number of elements used in its construction. This means that the network which has decidedly worse structure is able to solve tasks more efficiently (if it was well taught) than the network with optimal structure yet badly trained. There are known experiments in which network's structure was created

randomly (deciding in the course of drawing which elements to connect and in what way) and despite that the network was capable of solving complex tasks.

Let's have a closer look on the consequences of this last sentence as they are quite interesting and important. If the network was able to achieve correct results although its structure was designed randomly, this means that teaching process could each time adjust network's parameters to required operations, being a consequence of realization of chosen algorithm, so that solving process ran correctly despite fully randomized network's structure. These experiments were performed for the first time by above mentioned Frank Rosenblatt in early seventies ('70s) were very effective – the researcher rolled a dice or drew lots and, depending on the results, connected certain elements of the network together or not. Structure created in such way was completely chaotic (see figure 2.23) yet still the network was able to solve tasks reasonably.

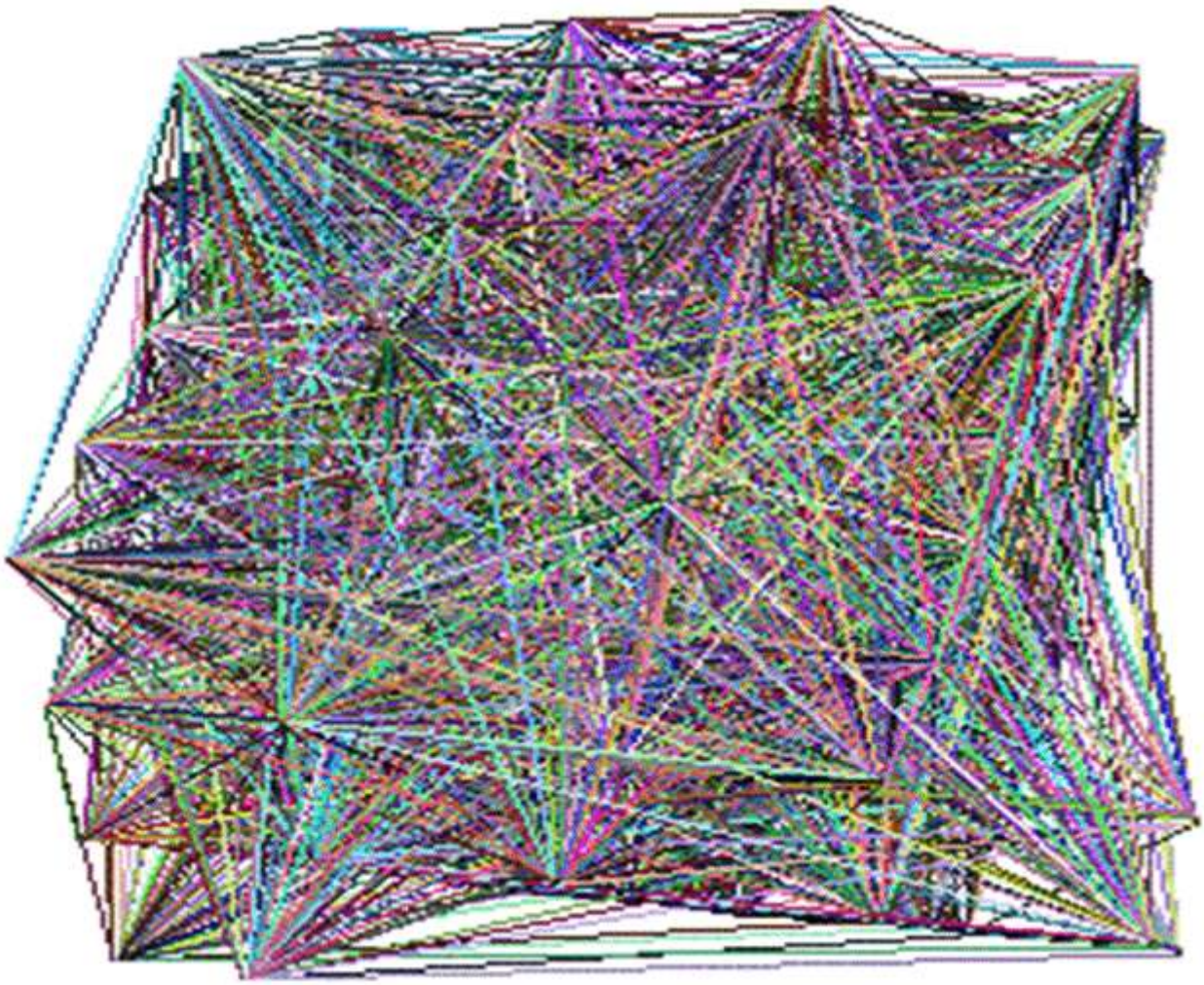


Fig. 2.23. Connection structure of the network which elements were connected to each other using the randomization rules. It is surprising that such network (after teaching process) is capable of maintaining purposeful and efficient tasks.

Results of these experiments reported by Rosenblatt in his publications were so astonishing that at first scientists did not believe that such thing is possible at all. However the experiments were repeated later (among others Russian Academy Member Glushkov constructed a neural network specially for this experiment and called it *Alpha*) and they proved that network with random connections can learn correct solving of the tasks, although of course the teaching process of such network is harder and takes more time than teaching the network which structure relates reasonably to the task that has to be solved.

Interesting thing is that philosophers were also interested in results reported by Rosenblatt, they claimed that this was a proof of certain theory proclaimed by Aristotle and later extended by Locke. It is a certain concept in philosophy called *tabula rasa* – a concept of a mind being born as a blank page which is filled in process of learning and gathering experiences. Rosenblatt proved that this concept is technically possible – at least in form of a neural network. Separate issue is the attempt to answer the question if it works with a mind of a particular man? Are, as Locke claimed, inborn abilities indeed nothing, and gained knowledge – everything?

We do not know this for certain, but it may not be this way really. But we certainly know that neural networks can gain all their knowledge only during learning and do not need to have created in advance, adjusted to the task structure. Of course network's structure has to be complex enough to enable „crystallization" of the needed connections and structures. Too small network will never learn anything as its „intellectual potential" does not allow that – yet the issue is not the structure itself but the number of elements. For example nobody teaches the rat the Relativity Theory although it can be trained to find the way inside complicated labyrinth. Similarly no-one is born already „programmed" to be a genius surgeon or only a bridge builder (this is decided by the choice of the studies) although some people's intellect can handle only loading sand on a truck with shovel yet still being supervised. It is just the way world is and no wise words about equality will not change it. Some individuals have enough intellectual resources others - do not, it is the same like with people being slender and well built and those looking as though they were spending too much time working on a computer.

In case of network situation is similar – one cannot make network having already inborn abilities, yet it is very easy to create a cybernetic moron which will never learn anything as he has too small abilities. Network's structure can be therefore of whatever kind as long as it is big enough. Soon we will learn that it also cannot be too large – as it is also unfortunate, but this we will discuss in a while.

2.6. How to choose a neural network structure wisely?

(Translation by Rafał Opiał, rafal.opial@op.pl)

Irrespectively of brought forward remarks showing that one can reach a goal even by teaching a non optimally chosen neural network to solve a problem, one have to formalize **some** neural network structure. And what's more, it is easy to reveal that choosing a **reasonable** structure that fits a problem requirements at the beginning, can significantly shorten learning time and improve its results. That is why I want to present you some remarks although I know that it will not be a solution for all kind of neural network construction problems. I feel indebted to give you few advices because we all know how difficult it is sometimes to chose **whichever** solution without clues. Placing a neural

network constructor in situation where he can freely adopt any structure he wants is similar to dilemma of abecedarian computer engineers that stare confused at system message:

Press any key...

Yeah, but which is **the any** key that I shall press?!

You may laugh about it but for me often similar is the question I hear from my students and Doctoral students: okay, but **what** is this **any** structure of neural network?

I'll say now few words about common neural network structures. One important thing is to remember that what comes below is not everything about possibilities and more – every researcher shall be a kind of Demiurge, a creator of new beings because neural networks with different structures are not completely understood and thus in this work we need every pair of... cerebral hemispheres.

I will start here with classification of commonly used neural network structures into two major classes: neural networks without feedback and with it. Neural networks belonging to first mentioned class are often called **feedforward** while the other in which signals can circuit for unlimited time are called **recurrent**.

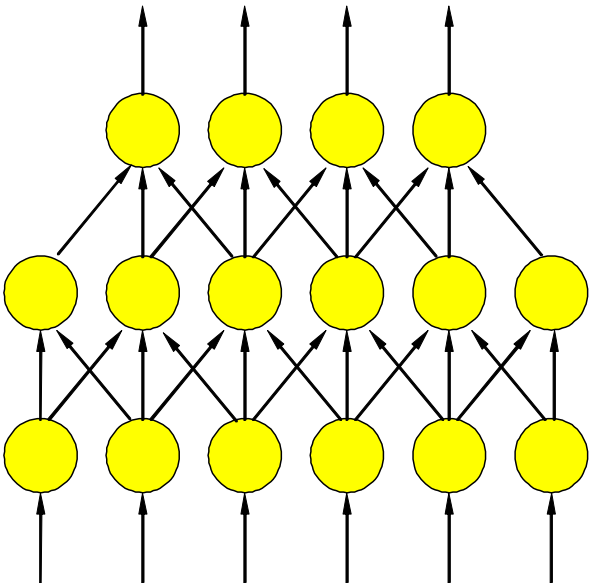


Fig. 2.24. Example structure of *feedforward* type neural network. Neurons represented by yellow circles are connected such way, which make possible transmission signals only from input to output.

- The feedforward networks are structures in which there is strictly determined direction of signal propagation. Signals go from defined input, where data about problem to solve is passed into neural network, to output where a network produces result (Fig. 2.24). This kind of networks is the most commonly used and the most useful. I will talk about them later in this chapter and in few that follow.

- The recurrent networks characterize occurrence of feedbacks (Fig. 2.25). In such networks signals can circuit between neurons for a very long time before it reach a fixed state, but there are also cases that this kind of neural network does not produce any fixed state.

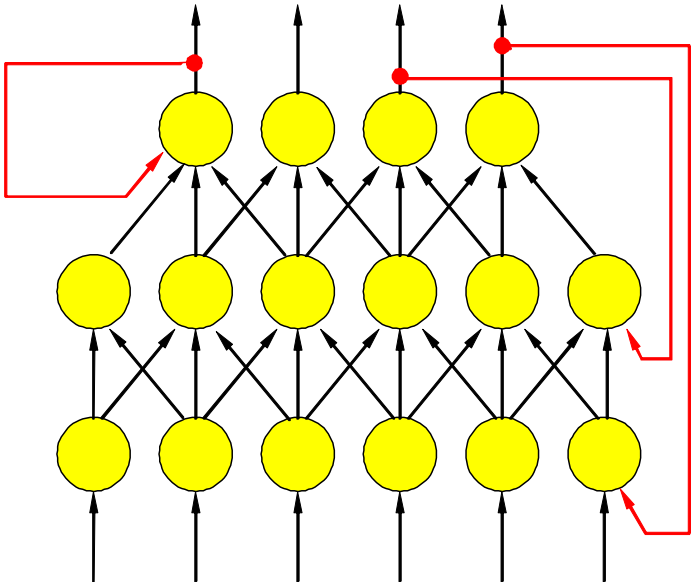


Fig. 2.25. Example structure of *recurrent* type neural network. Connections presented as red arrows are feedbacks, causing network to be recurrent one.

- Study of recurrent networks properties and abilities is more complex that it comes with feedforward networks, but on the other hand their computational potentials are astonishingly different than of other types of neural networks. For instance they can solve optimization problems – that is searching for the best possible solution, this is almost impossible to do for feedforward networks.

- Among all recurrent networks a special place belongs to these named after John Hopfield. In Hopfield networks the one and only kind of connection between neurons is feedback (Fig. 2.26).

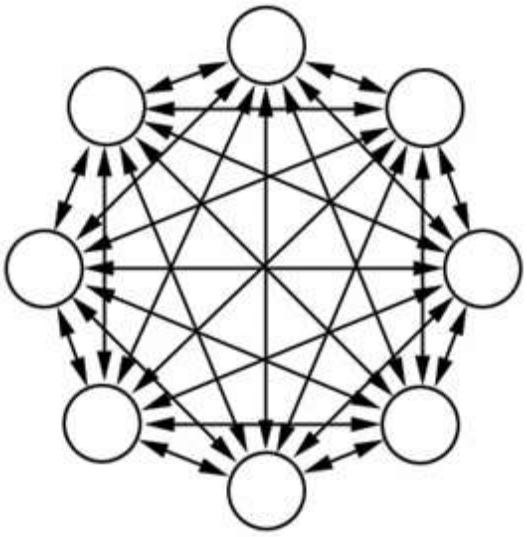


Fig. 2.26. Hopfield neural network, in which all connections are of feedback type.

Some time ago a true sensation was that a Hopfield network has solved a famous travelling salesman problem. That circumstance opened a way for Hopfield networks to manage important NP-complete class of computational problems, but this is something I am going to tell you later on. Despite this sensational breakthrough Hopfield networks did not become as popular as other types of neural networks so we will tell more about them just later, in chapter 11 of this book.

Since building a neural network with feedbacks is much more difficult than feedforward net and also controlling a network in which there is a lot of simultaneous dynamic processes is also much more difficult than controlling a network where signals politely and calm go from input to output, we start with one directional signal flow networks and then will slowly pass to recurrent nets. If you heard before about the most famous recurrent neural network, a Hopfield network, and want to know it better, you may pass over these chapters and start lecture of chapter 11, or you may (what I definitely recommend) arm yourself with patience and successively read one chapter after another.

If we focus on feedforward networks, we may state that the best and commonly used way to characterize their structure is a layer model. In this model we assume that neurons are clustered in sets called layers. Major interlinks exist between neurons belonging to adjacent layers. This kind of structure was already discussed in this chapter but it is worth to take a look at it again (Fig. 2.27).

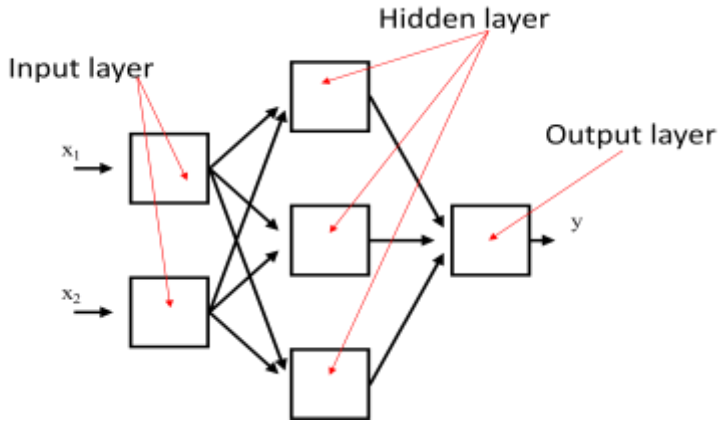


Fig. 2.27. Layered structure of the simplest neural network

I mentioned this in previous chapter, but it is worth to say again that links between neurons from adjacent layers may be constructed in many different manners (as it wish a constructor), although the most commonly used is all to all linkage, because we can count on, that learning process will lead to automatic cut off unneeded connections by setting their coefficients (weights) to zero.

2.7. What are optimal sources for 'feeding' neural networks?

(Translation by Krzysztof Królczyk; scoorviel@go2.pl)

Among all layers building/composing neural net we should begin with „input layer”. It’s purpose is, to collect data outside the network – this way we give an input – tasks needed to be worked on, problems to solve. Projecting this one, is likely to be easier than rest of the job – number of elements on this layer is strictly determined by amount of incoming data, which are being analyzed during this particular problem. However, sometimes even decision – which, how many, and what data should we feed to neural net is quite a dilemma. For example – we’d like to predict how will stock indexes

behave; it's well known that some researchers achieve encouraging results, what surely benefits in increased income for some, willing to take a risk, selling or buying on stock market according to output produced by neural net. But yet, publications explaining which data were put as an input, are rare and undefined...that is, of course quite self-explanatory. Mainly, it's easy to discover – neural net have been applied, learned (often we've access to learning routines), frequently – we're given outcome, cute-looking graphs showing how well had net predicted stock changes...it's always surprisingly slipping authors minds to write something more than, "network was given earlier achieved stock records, and financials analysis...". Obviously, whole preparation that's been made before feeding network, and their actual usage is interesting, yet unwilling to be discovered, secret.

Another thing, that's important is subtle. Numeric input, and output usually is limited. Most implementations assumption, all neurons input to be digits from 0 to 1 (or – what's more adequate - from -1 to 1), so if we require results from a different section – we'll need **scaling** (Fig. 2.28)**.**

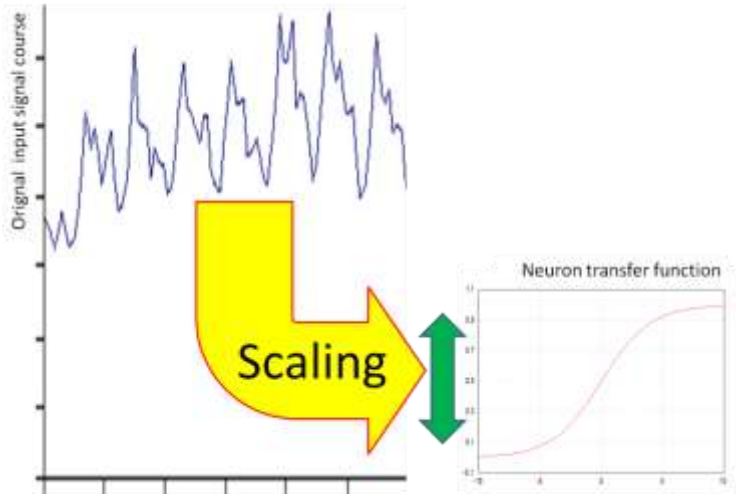


Fig. 2.28. Scaling of original input signal for proper representation in neural network

Problem with scaling inputs is actually less important, than dealing with our results (which we will see in next sub chapter), since they could be forced to take any signal, with any value. However, outcome is defined by neuron's characteristic, since it can't produce different signal than its „programming". To preserve unitary interpretation of all signals in neural network, and according importance weights, we usually scale output data, moreover it benefits in **normalization** input variables.

It might sound too difficult, when in fact it is, to guarantee network equivalency of her input signals. It's main problem is, that value of some of variables, being important to solution, are "naturally" very small, whereas others, not necessarily more important – produce high value. I.e. neural net used by doctor, helping, or at least trying to help him diagnose patient well, could be given amount of erythrocytes(due to blood analysis), and as another input – our object's heat. Both are equally important, and whether doctor's be right or wrong, could be influenced by one variable once, and another – some other time. However human's body temperature is generally low number (as You surely know 36,6 degrees Celsius for a healthy man), and even its slight changes – up, down, can be indicate serious health issues. On the other hand, amount of red cells (usually in one blood millimeter) is about 5 millions, and its difference(even if it's, say, a million) is nothing to be extraordinary afraid of. Didn't we do scaling before, neuron seeing two inputs at different levels

would neglect importance of temperature, ignoring blood analysis. Concluding – normalization allows us to treat every input according not to its value, but to its importance defined by neural net creator.

2.8. How to explain a network where does cow come from?

(Translated by Rafał Opiał rafal.opial@op.pl)

The next problem is more difficult. Data provided to the neural network (or gained from it as a result of solving a problem) not always have a numerical nature, which causes serious complications. Unfortunately, our world is set as not everything can be measured and presented in numerical way. Much data, which we want to use as an input or output from a neural network have a qualitative, a descriptive, or – as it is the most often called – nominal nature. That means that their values are certain **names** instead of **numbers**. A good way to explain that would be an example. Imagine you have a task to solve in which a neural network distinguish, if certain animals may or may not be dangerous to a human being (soon you will be solving similar problems with programs I published for you on the Internet).

Now, this task may need as an input, let's say, information about which part of the world does the animal come from. For if we know that the animal is big, has horns and gives milk, it is cow. But what determines, if it may be dangerous is the continent on which it live. European and Asian cows are calm and mellow as a rule, while some American cows, reared on open pastures, happen to be dangerous. Therefore in order to distinguish if an animal may or may not be dangerous, there is necessarily needed value of a variable determining animals origin. And here a handicap comes up: it is known how continents are named, but how to set a name "*Asia*" or "*America*" at the input of a neural network?

A solution for the put problem is usage of representation called "*one of N*", where N stands for the number of different possible values (names), that the nominal variable may adopt. On the figure 2.29 I have shown a coding manner for the method "one of N" where N = 3. The principle is simple and consists in using for every nominal variable that many neurons in input layer, as many different values particular variable may adopt (which is simply N). If, for example, we assume that identified animal may come only from Asia, America and Europe, then for representing the *origin* variable we must designate a set of three neurons. When we have done that and then we want to inform a network, that in particular case the value of *origin* variable is name *America*, we shall pass signal of value 0 to first input, value 1 to second input, and once again zero to the third input.

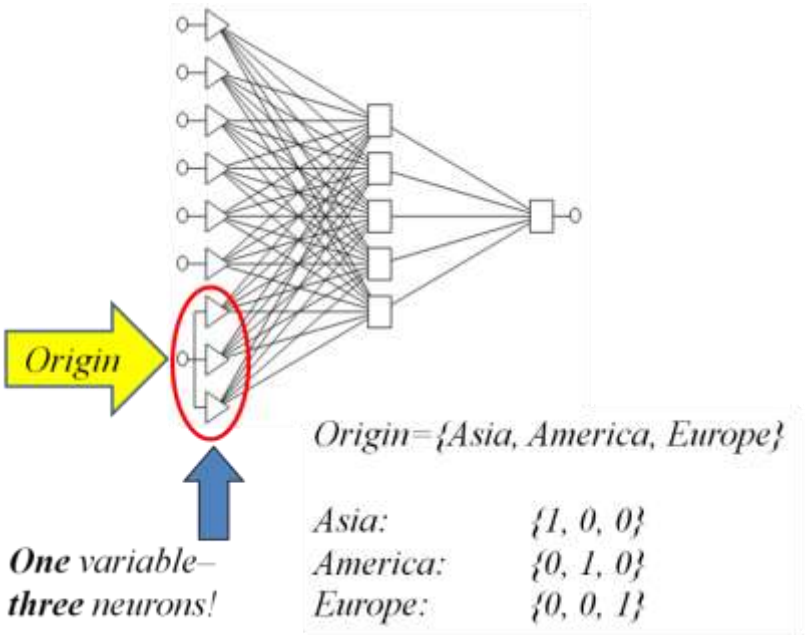


Fig. 2.29. Method of nominal input data coding

I think that since you are reading this book you are smart and ingenious person, so surely while reading about "one of N" method you were thinking with superiority:

*Why complicate things? **I** have a better idea! Let's code that Asia is 1, America is 2 and Europe is 3 and let's put values to **one** input of the network! Or if take into consideration that signal on the input shall assume values from range 0 to 1, let it be: Asia is 0, America is 0,5 and Europe is 1. Why nobody before got this idea?*

Well solution (hypothetical) you proposed unfortunately is not good.

Neural networks are very susceptible for **mutual relations** of values shown to them. If you adopt the first coding rule, neural network while learning will try to utilize fact that Europe is three times more than Asia – and of course it will be all nonsense. Even worse it will be in case of data having scaled character, because it will seem that *America* can be converted to *Europe* (by multiplying by 2), but *Asia* cannot be converted this way, because zero multiplied by whatever always stays zero.

In brief you **must** agree that nominal values have to be represented with "one of N" technique, although unfortunately it increases number of inputs for a network and causes increase of connections between the input layer and further layers of a network. Especially the second fact is inconvenient, because you have to remember that with every connection in a network is related a weight coefficient which value must be determined in the course of teaching, so additional inputs (and connections) – are additional trouble at teaching. Though in spite of these troubles there is no other (better) solution and you have to (I say it again) multiply inputs related to every nominal variable by using "one of N" schema – well unless you make up a really effective and better method.

Before I end this plot I have one more thing to add. Now usually there are exceptions from a certain rule. In case of "one of N" rule the exceptional case is when N amounts to 2. For the following "binary nominal variables" which for instance may be variable *Gender*, one can exceptionally code e.g. *Man = 0* and *Woman = 1* (or inversely) – and a network will cope with it. But this kind of binary

data is in fact “an exception that proves the rule” - and the rule is that nominal data **should be** coded with the “*one of N*” method.

2.9. How to interpret answer produced through net?

(Translation by Marcin Krasiński; mkk3@poczta.onet.pl)

The second characteristic layer which we will discuss in this subsection **is an output layer**, producing final solutions to the considered problem. These solutions are sent out as output signals from the whole network, so their interpretation is important for us, because we have to know and understand, what this network wants to inform us about. As for the number of exit neurons, it is simpler here, than in the case of entrance signals, because we usually know how many and what kind, of solutions we need and we have not dilemmas of the type - to add the signal or not , which was a difficult question in the process of designing the entrance layer of the net. As for concerns regarding the coding, method the situation on exit of network is similar to the one at the input layer. It means the numerical variables should be scaled to allow the neurons at the output layer to produce a number which represents the correct solution to the problem. In turn, the “nominal” variables should be presented at the output using the method “one from N”. However, on the exit of the net we expect a few problems typical of this layer, therefore we discuss then in this subsection.

The first problem is directly connected with the using of the "*One from N*" method. You remember that in this method the signal with maximum value (usually is 1) can have only one neuron on exit - this attributed to proper name, being value of a nominal variable. All remaining neurons from representing one variable group should have values equal to zero. This ideal situation when the nominal variable at the output layer is correctly indicated is shown in Figure 2.30.Where a convention is used, that the task of example considered network is (in some sense) reversal of task from Figure 2.29 – there it was need say to state on entry in which continent lives the classified animal (in order to execute classification, or it is dangerous, or not), and animal on entry be showed[1] here, and network has to give information which continent it lives in

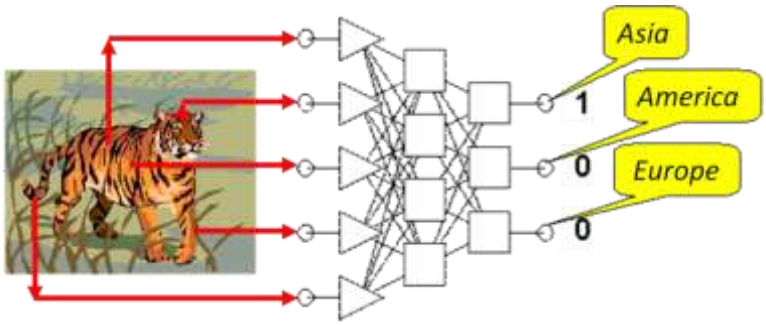


Fig. 2.30. Ideal situation on the output of network producing nominal variable

[1] Form of "*showing*" to the network of the animal depends on chosen information about his appearance and the building of body will send to neurons of entrance layer (shape of head, what color of fur, what paw, what tail etc.).This symbolizes red arrows on drawing, joining chosen elements of animal's building with entrance neurons to which information will be delivered on their subject.

The situation which is illustrated in Figure 2.30 happens only in theory (in practice it could happen only as a result of a favorable coincidence). In practice, taking into account limited precision of the computations done by the networks, we will talk about it below, almost always on exits of all neurons, which create a group attributed to some exit nominal variable, appear some non-zero signals. This situation is showed in Figure 2.31. A question arises: what to do with this?

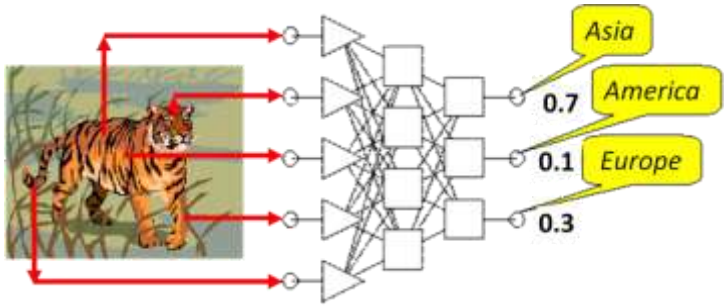


Fig. 2.31. Real situation, which can happened when working with nominal variable as network output

To obtain clear and unequivocal answers in such situations we should admit some additional criteria useful in further elaboration of the results at the output of the network. These additional criteria in processing of the results are the following: a threshold of acceptance and a threshold of rejection. Such additional treatment of the output signal leads to so called post-processing of the results. The exit signals computed by the neurons belonging to output layer of the network are, according to this concept, quantified by comparison of their current values with thresholds (as it is seen in Figure 2.32).

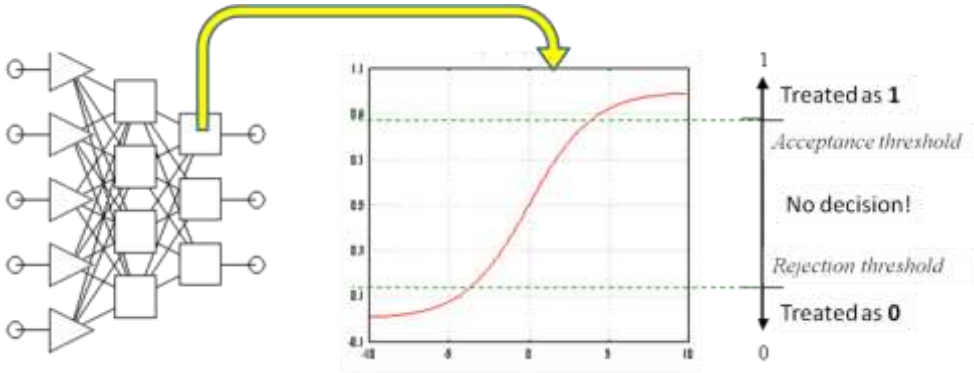


Fig. 2.32. Result of output signals post-processing, causing unambiguous result values despite inaccurate values on the neurons outputs

The value of parameters which are the threshold of acceptance and threshold of rejection can be chosen up to our needs, similarly, to decision rules responsible for neurons giving a lack signal. Experience shows , that it is better to set high requirements for network. It means that does not try "by force" to determine unequivocal value of exit variable nominal type, for example in a situation of such low clarity as it is seen in Figure 2.33.

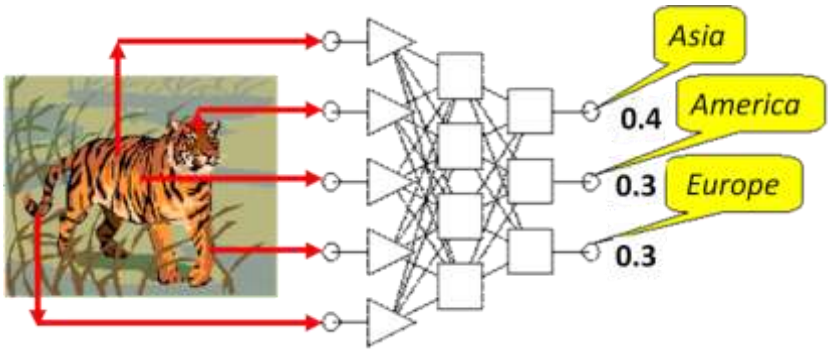


Fig. 2.33. Example situation, when only proper answer of the network is: “identification not possible”

It is always better to admit that the network is not able to achieve an unequivocal classification of the input signal than to accept a network decision which is probably false.

2.10 What is better to obtain from the network - number or a decision?

(Translation by Łukasz Brodziak; lukasz.brodziak@gmail.com)

While using a neural network one has to remember that results returned by the network even if they are numeric values (which can be subject to appropriate scaling) always are approximate. Quality of the approximation can vary, yet the accuracy of significant figures is out of the question, it is good if a product returned by a neuron has accuracy higher than two significant figures (which means that the fault can reach several percent). That is simply what the nature of the tool is. Awareness of the occurrence of such limitations is forcing an appropriate interpretation of the output signals, so that they can be used in sensible way, and induces thinking about which neural calculations model You want to use. Generally speaking, neural networks can form to types of models: regressive and classification model. Regressive model is a type of model in which, on the output of the network, we expect (and demand) specific numeric value, which is a solution of the problem. Interpretation of such model is presented on figure 2.34.

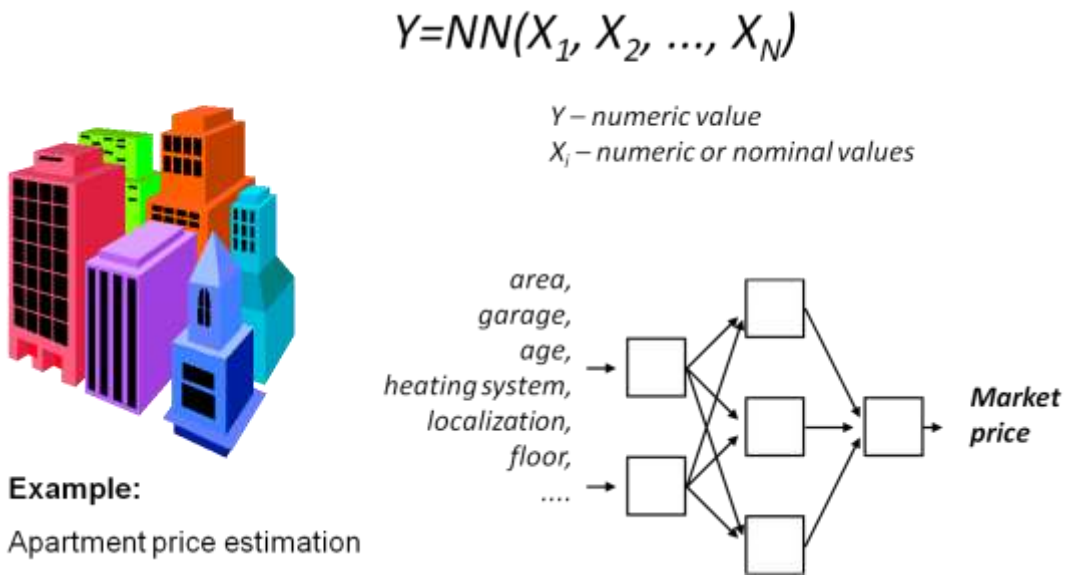


Fig. 2.34. Example of neural network regressive model

In the model shown on the above figure the task of the neural network is to estimate the price of the apartment. On the input we load data that can be numeric values (such as area in square meters) and data that are nominal values (for example if the apartment has a garage assigned to it), on the other hand on the output we expect a numeric value which would show the amount which can be gained on sale. As all apartment buyers and sellers know the market price of the apartment depends on many factors, yet nobody can present strict economic rules which would allow to predict that one apartment will cost this much, and the other one will cost twice as much. Seemingly it is impossible to predict because the price is each time a result of an unconstrained market game and sovereign decisions made by people who buy and sell a specific apartment. Still it appears that a neural network after long training (based on the previously finalized buy-sell transactions data) can form a regressive model of the problem so good that real prices obtained in successive transactions were only few percent different from the ones „*predicted*" by the network.

The alternative classification model is connected with a demand of obtaining from the network information on classifying an object described by input data to one of the classes. Of course in this type of problem on the output of the network a nominal variable will be placed (see figure 2.35).

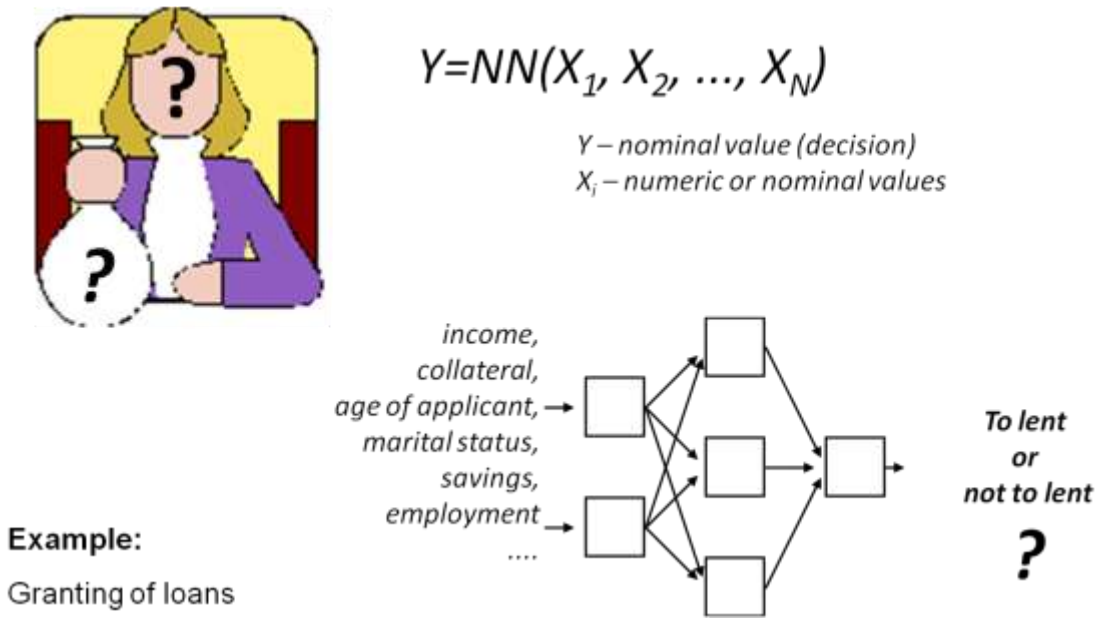


Fig. 2.35. Example of neural network classification model.

Example shown on the figure regards the problem of classification of the people (or companies) soliciting for a bank loan. Such client has to be classified by the bank worker to one of two categories: First one contains clients who are trustworthy and reliable, and therefore can get a loan because they are going to pay it back along with the interests and will give the bank profits. To the second category are numbered all the frauds and potential bankrupts who will not pay the loan exposing the bank to losses. How one can distinguish one category from the other? There are no strict rules or algorithms but correct answer can be provided by neural network taught with data from the past. There is usually a lot of such data as very bank has given a lot of loans and has full data of its clients both those who paid the loans back and those who did not.

Years of experiences in neural network exploitations proved that it is most convenient to interpret the tasks given to a neurocomputer in a way that would allow the answer to be given using classification model. For example we can demand that the network will state whether profitability of the investment is „*low*", „*average*" or *„big"* relatively whether the loan-taker is *„reliable", „risky"* or „*totally unreliable*". However the demand of specific estimation of the amount of profit, accuracy of the risk level or the limit of the amount that can be loaned to somebody will inevitably lead to frustration because in general it is the thing which network cannot do.

That is why the number of the outputs of the created network is usually bigger than number of questions that we seek an answer to. This happens because for many output signals we have to artificially add few neurons servicing each output – for example in a way that would allow the predicted range of output signal values to be divided into certain sub-ranges which distinction is essential for the user. In such case we should not force the network to return specific „guessed" value on its output instead we work in a way that each output neuron is responsible for **signalizing appurtenance of current answer to specific range** and usually that is all we need. Such classification network is way easier to build and teach, while making a network from which we „squeeze out" specific answers to mathematical problems is typical *„art for an art's sake"* – time-consuming fun with minimal practical application. Above formulated conclusions are summed up by figure 2.36

which presents one of the „classic" problems presented to neural network: prediction of currency rate.

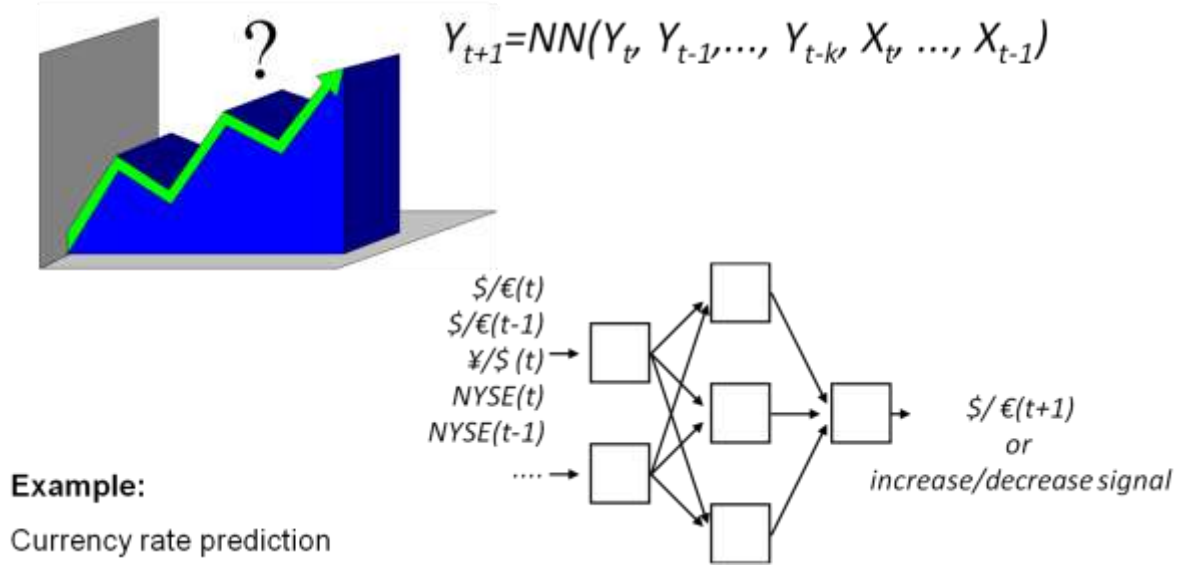


Fig. 2.36. Currency rate estimation is one of many problems in solving which the creator of a neural network can choose between regressive and classification model.

As shown on the figure the problem can be solved in two ways: Either one can build a prognostic model which attempts to determine how many Euros will a dollar be worth tomorrow or one can be satisfied with a model which only will signalize weather one should expect a raise or fall of the rate value on the next day. The second task will be easier for the network to solve and the prediction of rate value raise or its fall can be useful for someone wanting to buy dollars before travelling abroad.

2.11. Is having one network with multiple outputs better than having multiple networks with one output each?

(Translation by Lukasz Brodziak lukasz.brodziak@gmail.com)

There is another issue connected to the output layer of a neural network which can be solved in two ways and the choice of the way is a free decision of the network creator. It is known that it is always possible to create a network which will have any amount of outputs – as many as much data we want to get on the output as a result of solving the problem. It is not always an optimal solution because the teaching process of multiple-output network has to lead, while estimating weight values inside the network, to specific compromises which always have negative effect on the result.

An example of above mentioned compromises can be a situation when while setting a problem for a specific neuron of the hidden layer one has to take into account (which shows during its weight factors estimation) what role will this neuron have in the calculation of the values of several output neurons to which it sends its output signal when it will be calculated. It may happen that the role of a given hidden neuron, optimal from point of view of calculating with its aid one given output signal from whole network, will be significantly different then its role optimal for some other output. In such case the teaching process will change the weight values in this hidden neuron adapting it each

time to a different role – this will have an effect in long and less successful teaching. That is why it is better to divide a complex problem and instead of one network with multiple outputs build several networks which will use the same set of input data but have separate hidden layers and single outputs.

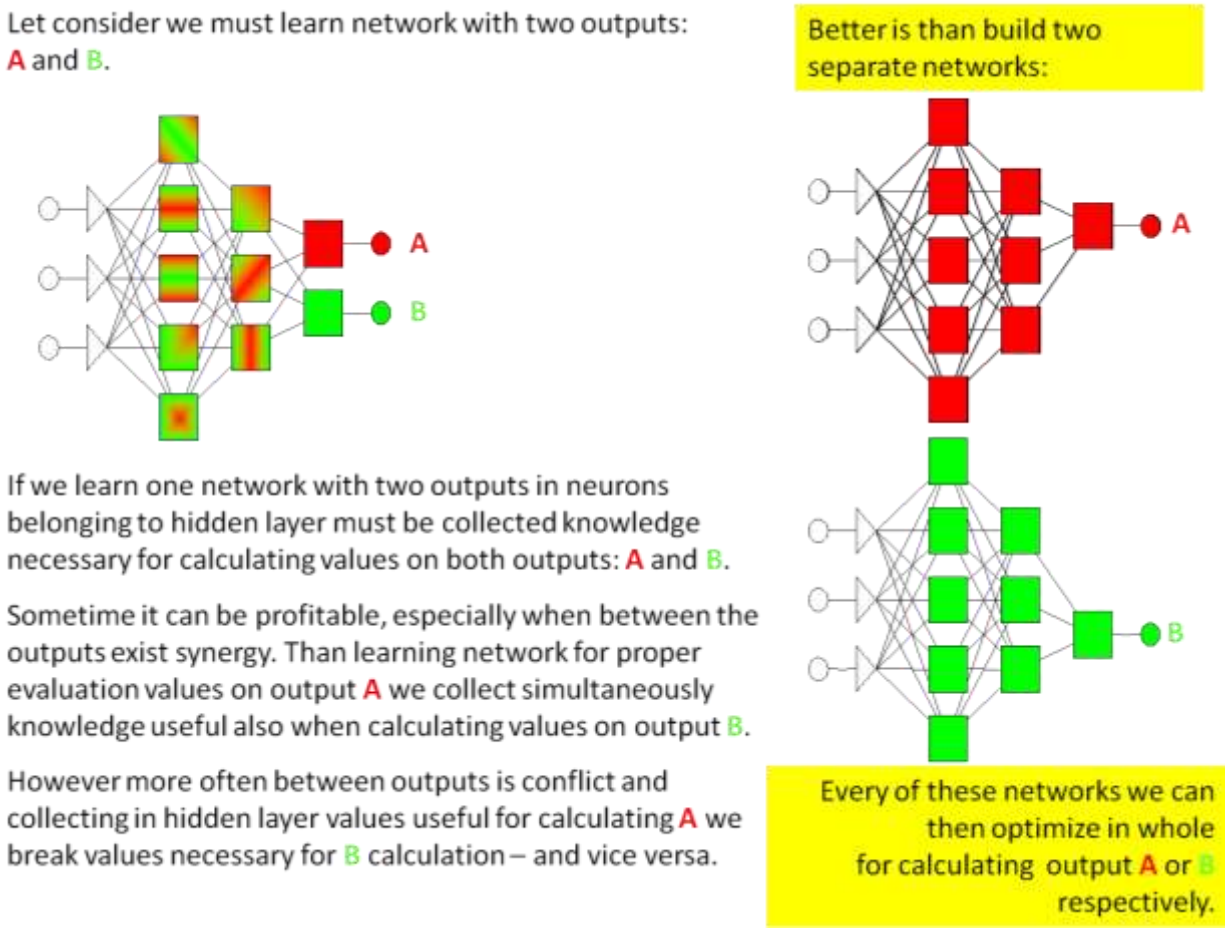


Fig. 2.37. Comparison of one network with two outputs versus two separate networks

The rule suggested in Figure 2.37 cannot be treated as a dogma because sometimes it appears that multiple-output network learns better than the one with single output. This, at first glance, paradox result can be substantiated with a fact that potential „conflict", described in paragraph above, connected to functioning of the hidden neurons and estimation of their roles while calculating several different output values may never occur. On the contrary sometimes during the teaching process one can observe an effect of specific synergy which consists in the fact that while setting (during the teaching process) the parameters of a hidden neuron which are optimal from the point of view of the „interests" of several output layer neurons, one can achieve success faster and more efficiently than while doing it in separate networks individually tuned for each output signal separately. That is why one should not assume that for sure one solution is better than the other, it is better to test them both and then making a choice. My own observation, based on experience in building hundreds of networks for various uses and on consulting tens of works by my students, shows that significantly more often the collection of networks with single output is an optimal solution – although biological neural networks are more often organized as multi-output aggregates.

You already know that every *feedforward* network has to have at least two of above mentioned layers – input and output. Yet, there are networks (see fig. 2.38) – they are called single-layered networks because they have only one learning layer. It is of course an output layer because as you already know the input layer of any network does not learn.

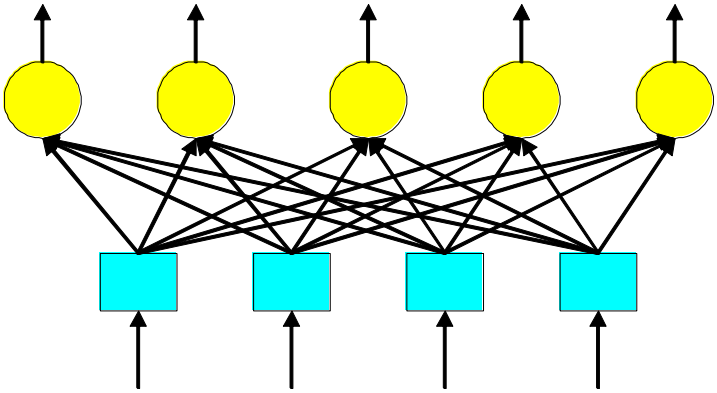


Fig. 2.38. Single-layered network

Yet still many networks (especially those solving more complex problems) must contain additional layers connecting input and output layer. Those layers put between input and output are called hidden layers. The name sounds quite mysterious that is why you may feel troubled while getting in touch with hidden layer problem for the first time. That is why I would like to explain to you what those layers are and in what sense they are „hidden".

2.12. What is hiding in hidden layers?

Not translated yet …

2.13. How many neurons do you need to get a well-working network?

(Translated by Anastasiya Zharkova, nastia_zhar@wp.pl)

It follows from the above remarks that the broadest possibilities for future use are offered by networks which have at least three-layer structure, with an input layer that receives signals, a hidden layer that elicits those characteristics of input signals that are needed, and an output layer which makes final decisions and provides a solution. Within this structure some elements are determined: the amount of input and output elements, as well as the principle of connecting successive layers. However, there are certain variable elements which you have to determine yourself. These are: **the number of hidden layers** (one or more) and **the number of elements in the hidden layer** (or layers) (fig. 2.40).

Despite many years of development of this technology, no precise theory of neural networks has yet been formulated, therefore these elements are usually chosen arbitrarily or by the process of trial and error. It may happen that the idea of a network's author about how many hidden neurons should be used and how they should be organized (e.g. as a one hidden layer or as several such layers) will not be quite correct. Nevertheless, it shouldn't have a **crucial** impact on the network's

operation, because during the learning process network always has a chance to correct possible errors of the structure by choosing appropriate connection parameters. Still, we must here specifically warn our readers about two types of mistakes which trap many researchers of neural networks (especially beginners).

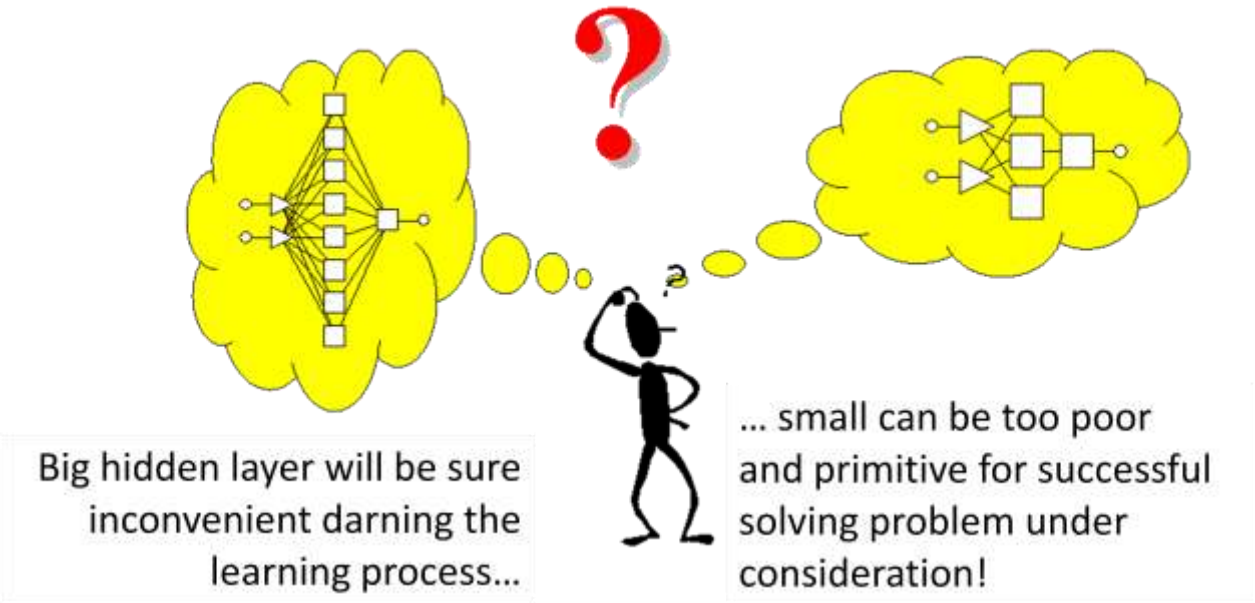


Fig. 2.40. Most important problem during neural network design is connected with number of hidden neurons

The first mistake consists in designing a network with too few elements – when there is no hidden layer or there are too few neurons, the learning process may fail miserably, since the network will not have any chances to imitate in its (too scanty) structure all the details and nuances of the problem that is being solved. Later, I will give you specific examples illustrating the fact that a network which is too small and too primitive cannot deal with certain tasks even if one teaches it very thoroughly and for a very long time. Simply, neural networks sometimes are like people: not all of them are talented enough to solve a particular problem. Luckily, there is always an easy way to check how intelligent a network is, because you see its structure and can count neurons, since the measure of a network's capability is merely the number of its hidden neurons. With humans it is more difficult!

Unfortunately, despite the freedom in building bigger or smaller networks, it sometimes happens that the intelligence of one's network is too low. This always results in a failure to use such a network for a specific purpose, because such 'neural dummy' with not enough hidden neurons will never succeed in the tasks it has been given, no matter how much you will toil at trying to teach it something.

However, network's intelligence shouldn't be also 'overdone'. The effect of network's **excessive** intelligence is not a greater capacity for dealing with problems that it has been set, but an astonishing phenomenon: instead of diligently acquiring useful knowledge the network begins to fool its teacher and, consequently, it doesn't learn at all! It may sound incredible at first, but it is true. A

network with too many hidden layers or too many elements in its hidden layers tends to simplify the task and, as a result, it 'cuts corners' whenever possible. In order for you to understand it, I have to explain briefly how network's learning process takes place (fig. 2.41).

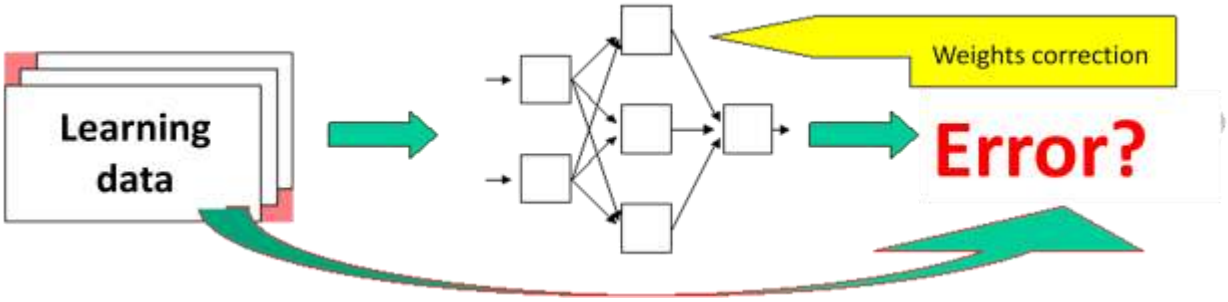


Fig. 2.41. Very simplified schema of neural network learning process

You will learn about the details of this process in one of the further chapters, but right now I must say that one teaches a network by providing it with input signals for which correct solutions are known because they are included in the **learning data**. For each given set of input data the network tries to offer its own output solution. Generally, the network's suggestions differ from correct solutions provided in the teaching data, so after comparing the solution worked out by the network with the **correct exemplar solution** in the teaching data, it becomes clear how big was the mistake made by the network. On the basis of mistake evaluation, the network's teaching algorithm changes the weights of all its neurons so that in the future the same mistakes will not be repeated.

The above model of learning process indicates that network aims at committing no mistakes when it is presented with teaching data. Therefore a network which learns well seeks such a rule for processing input signals that would allow it to arrive at correct solutions. When a network discovers this rule, it can perform tasks from the teaching data it has been provided with, as well as other similar tasks which it will be given during ordinary exploitation. We say then that a network demonstrates an ability to learn and to generalize learning results, and we treat it as a success.

Unfortunately, a network which is too intelligent, that is one which has a considerable memory in the form of a large number of hidden neurons (together with their adjustable weight sets), can easily avoid mistakes during learning process by memorizing the whole collection of teaching data. It then achieves great success in learning within an astonishingly short time, because it knows and gives correct answer for every question. However, in this method of 'photographing' the teaching data, a network which is learning from provided examples of correct solutions, makes no attempt at generalizing acquired information, and tries instead to achieve success by meticulously memorizing rules like "this input implies this output".

Such incorrect operation of a network manifests itself in the fact that it quickly and thoroughly learns the whole of the so-called teaching sequence (i.e. the set of examples used to show a network how it should perform tasks that it is given), but it fails embarrassingly in the first **test**, that is in a task from similar class but slightly different from the tasks presented in the process of learning. For instance, teaching a network to recognize letters brings immediate success (the network recognizes all letters it is shown), but the network fails completely to recognize a letter in a different handwriting or font (all outputs are zero), or recognizes them incorrectly. In such cases a closer analysis of network's

knowledge reveals that the network has memorized many simple rules like “if here two pixels are lit, and there are five zeros, letter A should be recognized”. Of course, such crude rules do not stand the test of a new task, and the networks falls short of our expectations.

The described symptom of ‘learning by heart’ is not displayed by networks with smaller hidden layer, because limited memory forces the network to do its best, and, using the few elements of its hidden layer, to work out such rules of processing input signal that would enable its correct use in more than one instance of required answer. In such cases the learning process is usually considerably slower and more tedious (examples needed by the network in order to learn have to be presented more times – often a few hundred or a few thousand times), though the final effect is usually much better. After a correctly conducted learning process has been finished, and a network works well with base learning examples, one has the right to suppose that it will also cope with similar (but not identical) tasks presented during a test. It is not always the case, but **it is often true**, and it must be the base for our expectations regarding the use of the network.

To sum up, it is important to memorize the following rule: do not expect miracles, so a presumption that an uncomplicated network, with few hidden neurons, will succeed in a complicated task, is rather unrealistic. However, too many hidden layers or too many neurons also lead to a significant decline in learning process. The optimal size of hidden layer lies somewhere between these two extremes.

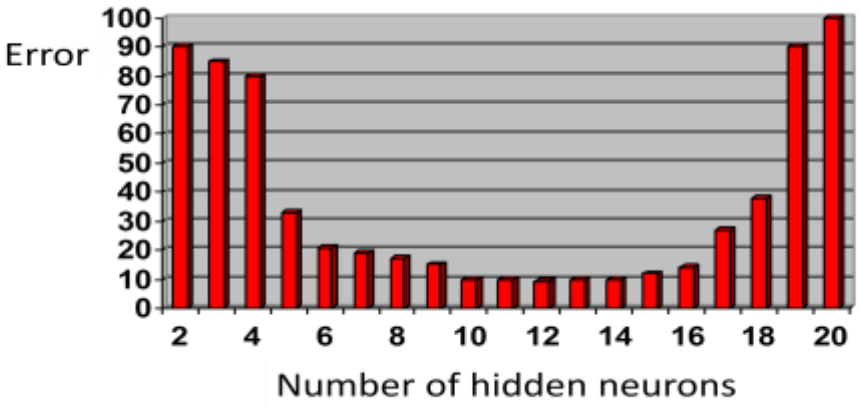


Fig. 2.42. Exemplary relation between hidden neurons and errors made by network

Figure 2.42 shows (on the basis of particular computer simulations) the relation between errors made by networks and the number of their hidden neurons. It proves that there are many networks which operate with almost the same efficiency, despite different number of hidden neurons – therefore it is not so difficult to hit such a broad target. Nevertheless, one must avoid the extremes (i.e. networks that are too big or too small). Especially harmful are additional (excessive) hidden layers, and it comes as no surprise that a network with fewer hidden layers often produces better results (because one can teach it more thoroughly) than a theoretically better network with more hidden layers, where the teaching process ‘gets stuck’ in the excess of details. Therefore one should use networks with one or (but only as an exception!) two hidden layers, and fight down the temptation to use networks with more hidden layers by fasting and cold baths.

2.14. Control questions and self-work tasks

Not translated yet …

3. Teaching the networks

3.1. Who is the tutor, who will teach the network?

(Translation by Agata Krawcewicz, hogcia@gmail.com)

A cycle of neural network's activity can be divided into stages of **learning**, when the network acquires information needed to determine, what and how should be done, and the stage of regular work (sometimes also called an exam), when, based on gained knowledge, the network has to solve particular **new** tasks (just because solving tasks, on which you based on when learning the network, is of no interest to you at all – you already **know** these solutions). The key to understanding how the networks work and its abilities is the learning process itself, so we are going to start getting to know the networks from writing about this exact process, whereas activities of already taught networks of various types are going to be discussed in the next chapter

Two variations of learning can be distinguished: **with a teacher** or **without a teacher**. We are going to talk about learning without a teacher separately in the next chapter, so meanwhile we will focus on learning with a tutor. Such learning is based on the fact that the network is given examples of correct actions, which it should later mimic in its current tasks (at the time of the exam). An **example** should be understood as a specific input and output signals given by a teacher, showing what is a required response of the network for a given configuration of the input data. The network watches, what is the connection between given input data and a required result (outcome, which should be produced as an output) and learns to imitate this rule.

Remember: while learning with a tutor you always have to deal with a pair of values – a sample input signal and a desired output that is a required response of the network for this input signal. Of course the network can have many inputs and many outputs, and in such case the mentioned **pair** means in fact two collections of values: a set of values, which is a complete set of input data and a set of values, which should appear on the output of the network, as a full solution for a task. But they always go together: **data** for a task and its **solution**.

At this moment, let's explain who is a teacher in the method of learning with a tutor. Under this name I do not necessarily mean a human being, who teaches network by himself – even though you will play with your networks. In practice a role of a tutor is taken over by the computer itself, who models the mentioned network, because it is easier and more comfortable. Unfortunately neural networks are not very smart and for good learning in a harder task, we need hundreds or sometimes even hundreds of thousands of steps! Who would have strength and patience to tutor such an ignorant!? That is why when talking about a **teacher** (tutor), I always mean a computer program supplied by a human with a so called **learning set**.

What is a learning set? Look at figure 3.1.

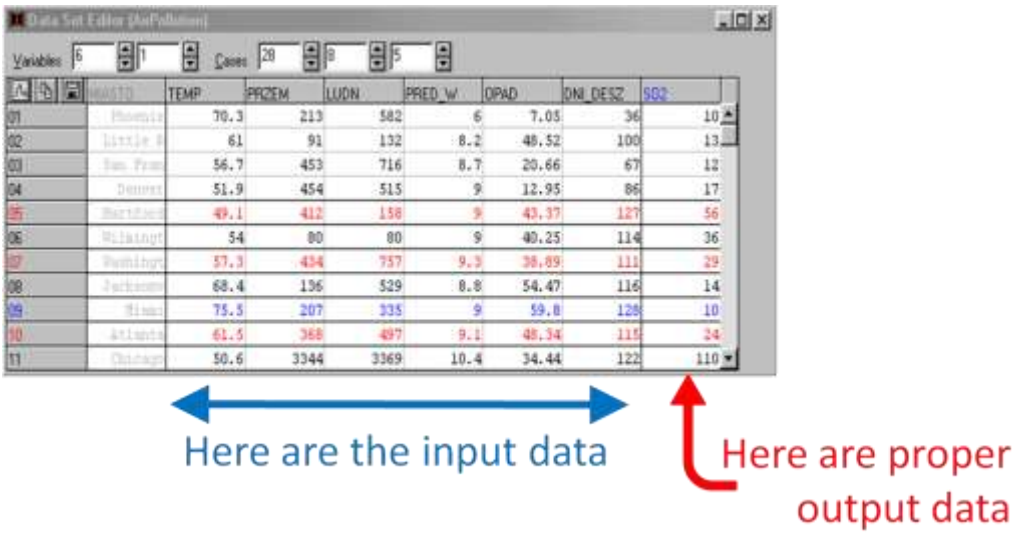


Fig. 3.1. Example of learning set

As you can see, there is a table with some sample data in the figure. These are the data concerning a pollution rate in various American cities, but it could be any other data. It is important that it is a real-life data, taken from a real database. I signalized that, leaving the elements of an original window (which comes from a program which was operating on this database) on figure 3.1. Among the data collected in the database we can isolate those, which will be used as outputs for the network (look at the range of columns of the table, which is marked by an arrow at the bottom of figure 3.1). In the example presented on this figure it will be data allowing us (perhaps) to foresee the level of air pollution, so it contains data about the city's population, it's industrialization, weather conditions etc. Based on those data, described as input data, the network will have to predict the average level of air pollution in every city.

For a city, in which pollution level has not been marked yet, we will have to guess it – and that is where we will use the network, which we will have earlier taught. Nevertheless, on the stage of learning the network you will use the fact, that in our database, used as a learning set, the pollution data for some cities is known – and it has been placed in an appropriate column of the table, which is marked with a red arrow on figure 3.1.

Therefore, you have exactly what you need to teach the network on figure 3.1: a set of data, in which the appropriate input and output data is given, and which make **pairs**: we can see reasons (population, industrialization, and weather conditions) and the result (value of air pollution). The network uses these data and will learn how to function properly (guessing the value of air pollution in cities, in which proper measurements have not been taken yet), using one of many strategies of learning that we know today. Exemplary learning strategies will be thoroughly discussed a little bit further. Whereas meanwhile please pay attention to another detail of figure 3.1: one column of the table is shown there with letters in gray – not properly black - so that it in fact can be seen, but barely. This type of color suggests that it is sort of less important. And so it is indeed: in this column there is data concerning names of particular cities. That information is required in the database, because based on it new data can be put and results given, but for a neural network such an information is useless (air pollution does not depend on the name of a considered city), so even though adequate data is available in the database, we are not going to use it for learning the

networks, There can be more of such unnecessary data (not used in the process of learning the network).

And so we should memorize that the tutor in the process of learning a network will usually be a collection of data, which though is not taken “as it is”, but is adjusted to function as a learning set with a proper configuration of data (you have to clearly state, which of them you will use as an input data, and which as an output data) and through its cautious selection. You should not “litter” a network with data, which you indeed have, but which you know (or suspect), that are not useful from the point of view of finding the solutions to the stated problem.

3.2. Can the network learn all by itself?

(Translation by Agata Krawcewicz, hogcia@gmail.com)

Close by to the previously described schema of the learning with a tutor is also a series of methods of the so called learning without a teacher (or self-learning networks) that are in use. These methods consist of passing only a series of example input data to the entries of networks, without giving any information concerning desirable or even only anticipated output signals. It seems that the properly designed neural network can use only observations of entrance-signals and build the sensible algorithm of its own activity on their base - most often relying on the fact, that classes of repeated (maybe with certain variety) input signals are automatically detected and the network learns (completely spontaneously, without any open learning) to recognize these typical patterns of signals.

With the self-learning networks you must also have the learning set – only that this set will only consist of data provided for input of the network. There is no output data given, because the technique of the learning without the teacher we should apply in the situation, when we do not know, what to demand from the network analysing some data. If we, for example, take data shown in fig. 3.1, to the learning without a teacher, you would only use columns described as input data, instead of giving the information from the column noted with the red pointer to the network. Network which would gain knowledge in such process of the learning would not have chances on predicting, in which city the air pollution will be greater, and wherein smaller, because it will not gain such knowledge itself. However analyzing the data on the subject of different cities the network may for example favor (just by itself!) a group of large industrial cities and learn to differentiate these cities from small country towns being centers of agricultural regions. This distinction will surrender to deducting from given input data on such a rule that industrial towns are mutually similar, whereas agricultural cities also have many common characteristics. On a similar rule the network can, only by itself, separate cities with beautiful and bad weather and make many other classifications, based only on values of observed input data.

Notice that the self-learning network is very interesting from the point of view of the analogy, which exist between suchlike activity of the networks and with the activity of the human brain: people also have an ability of spontaneous classifying of encountered objects and phenomena (some call this "a formation of notions"), and after the execution of the suitable classification, recognizes another objects as adherent to one of these earlier recognized classes. Self-learning is also very interesting from the point of view of its uses, because it demands no openly given external knowledge - which can be inaccessible or gathering which can be too troubling - the network will accumulate all

necessary information and pieces of news all by itself. In one of the further chapters I will describe very exactly (and I will show demonstratively across suitable programs!) what is this self-learning of the networks based on.

Now you can imagine (rather for fun and stimulation of the imagination, than from a real need) that a self-learning network with TV camera can be sent in the unmanned space probe on Mars. We do not know, what are the conditions on Mars, so we do not know, which objects our probe should recognize. What is more - we do not even know how many classes of objects will appear! It does not matter, the network will cope with it itself (look at fig. 3.2). The probe lands and the network begins the process of self-learning. At the beginning it recognizes nothing, only observes. However, over time the process of the spontaneous self-organization will lead to a situation that the network will learn to detect and to differentiate between different types of input signals which appear on its entry: separately rocks and stones, separately plant forms (if they will be there), and separately living organisms. If we will give the network sufficient amount of time - it can be so educated that it can differentiate Martian-men from Martian-women - though her creator did not even know that they exist!

Fig. 3.2. Hypothetic planetary lander powered with self-learning neural network can discover not known forms of life (“alien”) on mysterious planets

Of course earlier described self-learning Mars lander is exclusively a hypothetical creation, even though networks that form and recognize different patterns exist and are eagerly used. For example we could be interested in the fact of how many and which forms of the certain little known disease can in fact be found. Is this one sickness unit, or several? What are they differed by? How to cure them?

It will be sufficient to take a self-learning neural network and show it the information on registered patients and their symptoms for a long enough period of time. Sometime later the network will give the information on how many typical groups of symptoms and signs will be detected and on the

ground of which criteria one can differentiate patients classified to different groups. Explorer then only has to name these groups with properly wise sounding Latin names of illnesses - and he can already run to the tailor, to try on the tailcoat sewn for the solemnity of the handing of Nobel Prize!

The described method of self-learning has of course (as everything on this world) definite defects - however, I will describe them a little later, when it will be already clear how all this works. Nevertheless, it has so many undeniable advantages, that one ought to be surprised with its comparatively small popularity!

3.3. Where and how do neural networks gather obtained information?

(Translation by Agata Krawcewicz, hogcia@gmail.com)

Let us let us take a closer look at the process of learning with a teacher. How does it happen that the network gains and gathers knowledge? Well, a key-notion here are the weights on the entrance of each neuron, which were described in the previous chapter. Let us remind: every neuron has many inputs, by means of which it takes signals from other neurons and input signals given to the network as data for its calculations. The parameters called weights are united with these entries; every input signal is first multiplied by the weight, and only later added up with other signals. If we change values of the weights - the neuron will begin to function in other way in the network, and as a result - the whole network will begin to work in an another way. The whole art of learning the network relies on the fact that we should choose weights in such a manner, that all neurons would perform the exact tasks, which the network demands from them.

There can be thousands neurons in the network, and every one of them can have hundreds of inputs - so it is impossible for all these inputs to define the necessary weights at one time and arbitrarily by oneself. One can however design and realize the process of learning relying on starting activity of network, with the certain random set of weights, and gradually improving them. In every step of the process of learning, the value of weights from one or several neurons undergoes a change, and the rules of these changes are made in the way, that every neuron is able to qualify, all by itself, which of its own weights it has to change, and which way (increase or decrease), and also how much. Of course when determining necessary changes of weights the neuron can use the information being descended from the teacher (as far as we use learning with a teacher), however, it does not change the fact that the process changing the weights itself (being the only memory trace in the network) runs in every neuron of the network spontaneously and independently, thanks to what it can be realized without the necessity of the first-hand stable supervision from the person supervising this process. What is more, the process of the learning one neuron is independent from, how any other neuron learns, so learning can be conducted simultaneously in all neurons of the network (of course under condition of constructing a suitable network as the electronic system, and not in a form of simulation program). It also allows us to reach very high speeds of learning and a surprisingly dynamic increase of "qualifications" of the network, which literally grows wiser and wiser in front of us!

I will once again stress, because it has a key-meaning: the teacher need not get into the details of the process of learning - it will be sufficient that the person will give the network an example of correct

solution. The network will compare its own solution, which it obtained from the example that we used, which originated from given learning set, with the solution which is recorded in the learning set as a model solution (so most probably correct). Algorithms of learning are constructed the way, that the knowledge about the value of the error which the network makes, is sufficient to correct values of its weights, whereat every neuron separately (controlled with the mentioned algorithm) corrects its own weights on all entries all by itself - if it only gets the message, what error was committed. Is very simple and the simultaneously efficient mechanism, shown symbolically in fig. 3.3. Its systematic use causes the network to perfect its own activity, till at last it can solve all assignments from the learning set and on the grounds of generalization of this knowledge - it can also solve other assignments which will be introduced to it on the stage of "examination".

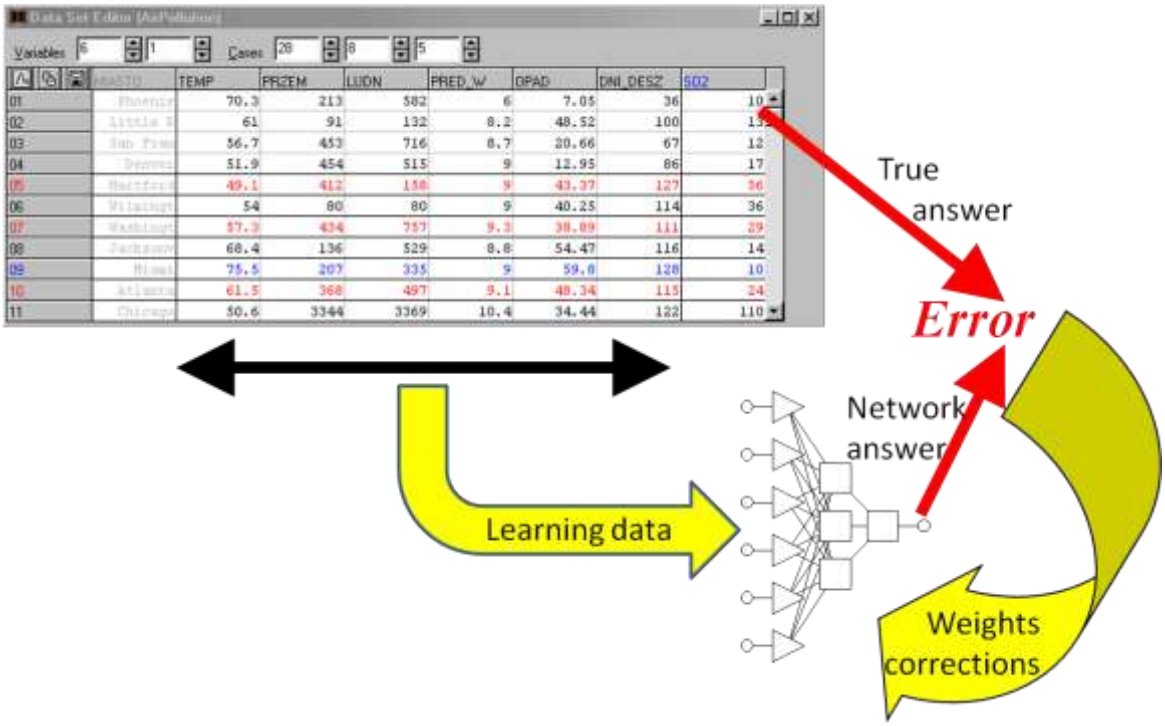


Fig. 3.3. Typical step of neural network learning

The manner of the learning of the network, described earlier, is used most often, but in some assignments (for example at the recognition of images) one need not give the network the exact value of the desired output signal, but for efficient learning it is sufficient to give the network only general information on the subject, whether its current behavior is correct, or not. At times one speaks directly about signals "of the prize" and "the punishment", on the ground of which all neurons of the network, all by themselves, find and introduce proper corrections to their own activity. This analogy to the training of animals is not quite accidental!

3.4. How to organize learning the networks?

(translation by Agata Krawcewicz, hogcia@gmail.com)

Necessary changes of values of weight coefficients in each neuron are counted basing on special rules (sometimes called the paradigms of networks), where the number of different rules that are used today and their varieties is truthfully extreme, because almost every explorer tried to carry his own contribution in to the field of neural networks in the form of a new rule of learning. One opinion about this problem gives a collection of algorithms of learning described in my book "Neural networks" to which I constantly send the more inquiring Readers for more detailed information (http://winntbg.bg.agh.edu.pl/skrypty/0001/). Here I will talk over shortly (without the use of mathematics, because such a deal we contracted at the very beginning of this book!) two basic rules of learning: the rule of the quickest fall, lying at grassroots of the most algorithms of learning with a teacher and the Hebb rule, defining the most simple example of learning without a teacher.

The rule of the quickest fall relies on the fact that every neuron having received definite signals on its entries (from the networks entries or from other neurons, which are earlier levels of processing the information) makes its own output signal using possessed knowledge in the form of earlier settled values of all amplification factors (weights) of all entries and (possibly) the threshold. Manners of marking the value of output signals by neurons, on the ground of input signals were talked over more precisely in the previous chapter. The value of the output signal, appointed by the neuron in the given step of the process of learning is compared with the standard-answer given by the teacher within the learning set. If there is a divergence (and at the beginning of the process of learning such divergence will almost always appear, because where on earth does this neuron have to know, what do we want from it?) - the neuron gets the difference between its own output signal and the value of the signal which would be - according to the teacher - correct, and also fixes (by means of the method of the quickest fall which I will soon explain), how to change the values of the weights, so that this error will most quickly grow smaller.

In further considerations it will be useful to know the notion of the area of the error which we will now introduce and talk over precisely. And so you already know the fact that the activity of the network relies on the values of weight coefficients of neurons being its elements. If you know the set of all weight coefficients, appearing in all neurons of the whole network, then you know how such network will act. Particularly you can show to such network (in turns) all examples of assignments accessible to you, together with solutions, which are a part of learning set. Every time the network will make its own answer to the asked question - you can compare it with the pattern of the correct answer which is found in the learning set, marking the error which the network committed. A measure of this error is usually the difference between the value of the result delivered by the network and with the value of the result read from the learning set. To overall rate the activity of networks with the defined set of weight- coefficients in its neurons - we usually use the pattern of the sum of squares of errors committed by the network for each case from learning set. Before summing up the errors are squared, to avoid the effect mutual compensating of positive and negative errors, and what is more the squaring causes that the especially „heavy penalty” meets the network for large errors (twice greater error is a quadruple component in the created sum).

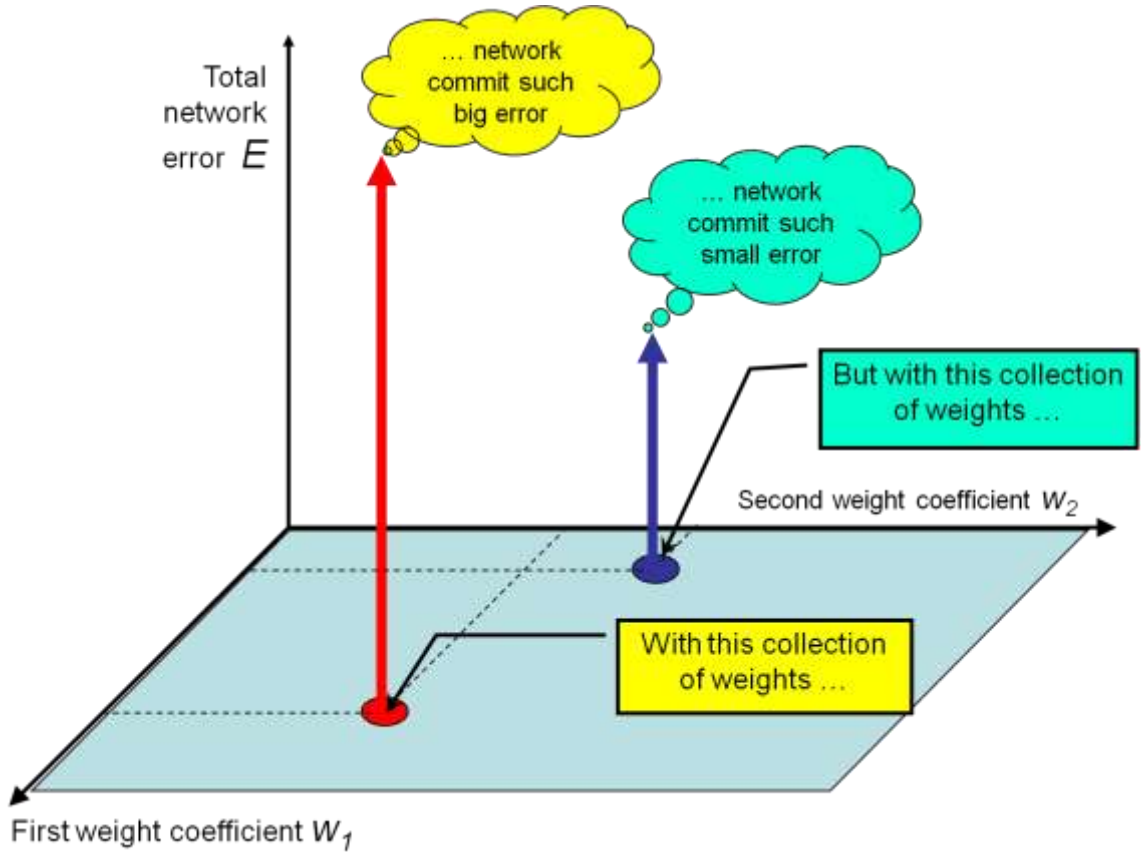


Fig. 3.4. Method of error surface forming. Discussion in the text. Please observe the yellow rectangle and yellow cloud fist and navy blue rectangle and cloud next.

Look at figure 3.4. A situation was shown on it, which could take place in a network so small that it would have only two coefficients of the weights. Such small neural networks do not exist, but imagine that you have such a network, just because only for such very small network will we succeed to draw its behavior without going into difficult, multidimensional spaces. Every state of better or worse learning of this network will be joined with some point on the horizontal (light-blue) visible surface shown in the figure, together with its coordinates, that is with both considered weights-coefficients. Imagine now that you placed such values of weights in the network, which comply with the location of the red point on the surface. Examining such network by means of all elements of the learning set, you will find the total value of the error of this network - and in the place of the red point you will put a (red!) arrow, pointing up, with the height being the calculated value of the error (according to description the vertical axis in fig.).

Next, choose other values of weights, marking other position of the point on the surface (navy blue) - and perform the same acts, receiving the navy blue pointer.

And now imagine that these acts you perform for all combinations of weight- coefficients, that is to say for all points of the light-blue surface. In one places errors will be greater, in other smaller, what you would be able to see (if you had a patience to examine your network so many times) in the form of error surface, spreading over the surface of the changed weights. An example of such surface I have shown you it in fig. 3.5.

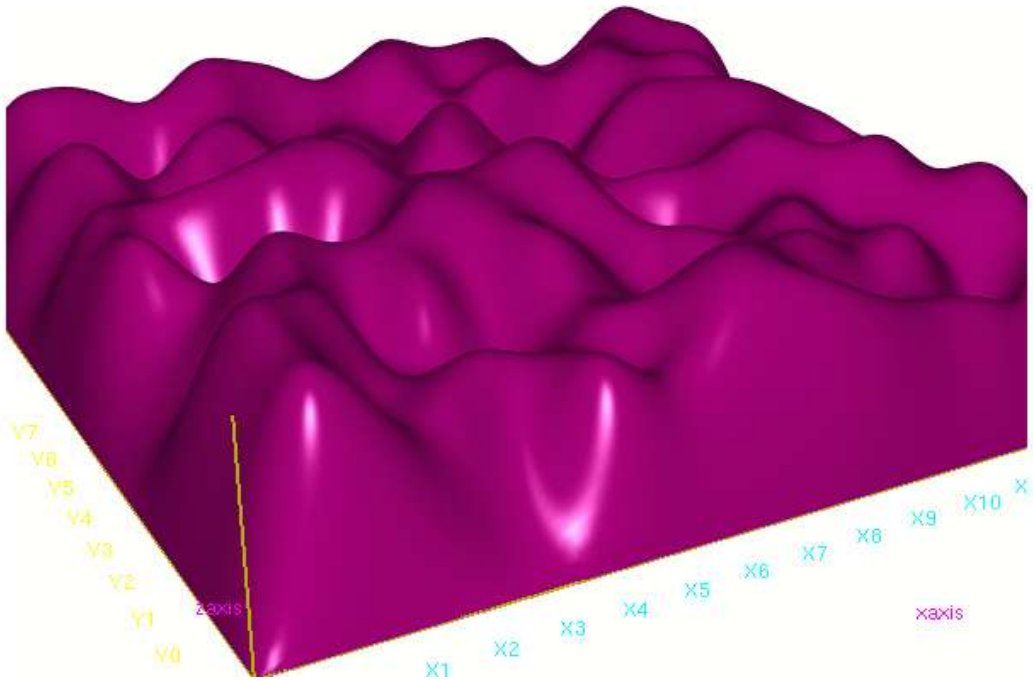


Fig. 3.5. Example of error surface formed during the neural network learning

As you can see there are many „knolls" on this surface - those are the places in which the network commits especially many errors, so such places should be avoided, and we have many deep valleys which we find very interesting, because there, on the bottom of such valleys, the neural network commits little errors, that is to say solves its assignments especially well.

How should we find such valley?

And so you should consider learning the neural network as the multistage-process. During this process you will try step by step to improve values of weights in the network, changing old (worse) sets of the weights, causing that the network to commit a large error, for new, regarding which you will be hoping (but you will not be certain) that they are better. Look at the figure 3.6, whereon I have tried to illustrate this.

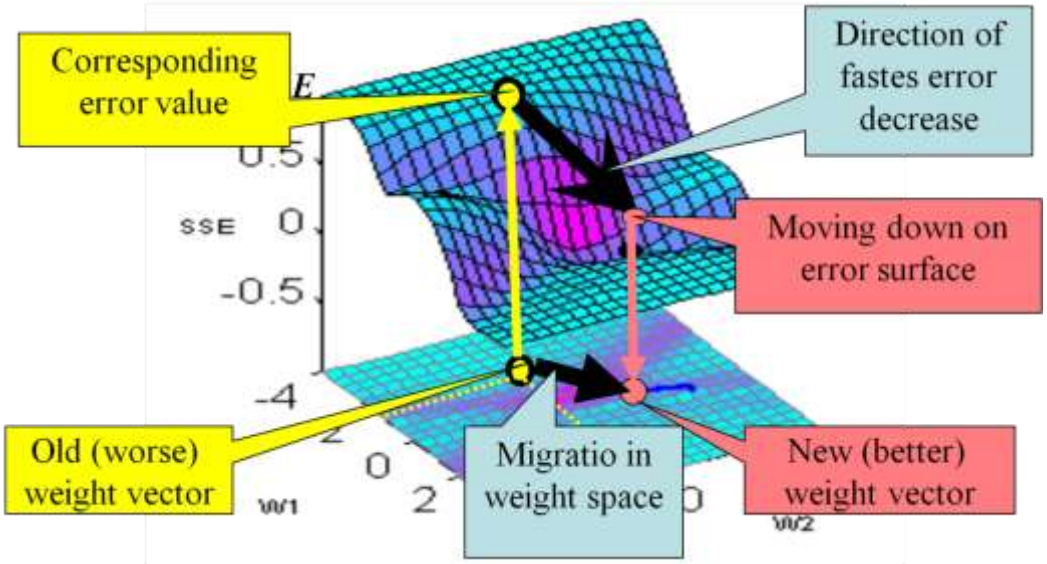


Fig. 3.6. Illustration of neural network learning process as “sliding down” on error surface

You begin from the situation shown in the left bottom corner of the figure, which means you have some old set (a vector) of weights, noted on the surface of parameters of the network with the yellow circle. For this vector of weights you find the error, which the network makes, and you „land” on the surface of the error in the place which is marked with a yellow arrow in the left upper corner of the figure. Nothing good can be said about this situation: the error is very high, so the network has temporarily very bad parameters. It is necessary to improve this.

How?

Well exactly. Methods of learning the neural networks can find which way it is necessary to change the weight coefficients, to obtain the effect of the diminution of the error. Such direction of the quickest fall of the error is noted in fig. 3.6 as a large black pointer. Unfortunately, the details of how the methods of learning do this, cannot be explained without using complicated mathematics and such notions as the gradient or the partial derivative, however conclusions from these quite complicated mathematical considerations are simple enough. And so every neuron in the network makes the modification of its own weight- (and possibly the threshold) coefficients, using the following two simple rules:

- weights are changed these more strongly, when a greater error was detected;
- weights connected with these entries, on which large values of input signals appeared, are changed more, than weights of these entries on which the input signal was small.

Previously described basic rules in practice still need several additional corrections (in a moment I will say more about them), however the described outline of the method of learning is surely clear. Knowing the error committed by the neuron and knowing its input signals you can easily foresee, how its weights will change. Also notice, how very logical and sensible are these mathematically led out rules: For example the faultless reaction of the neuron on given input signal should of course cause leaving its weights without a change - because they led us to a success. And this is just what is happening!

Notice that the network using described methods in practice, breaks the process of learning itself, when it is already well trained, because small errors cause only minimum, "cosmetic" corrections of the weights. It is logical, similarly as the rule of the subordination of the size of the correction from the size of the input-signal delivered by considered weight - because these entries on which greater signals appeared had a greater influence on the result of the activity of the neuron which proved to be incorrect, so it is necessary to "sharpen" them more strongly. Particularly the described algorithm causes that for the entries on which in this very moment the signals were not given (during calculations they had zero-values), suitable weights are not changed, because not we do not know whether they are good or not, because it did not participate in the creation of the current (surely incorrect, if it is necessary improve it) output signal.

Returning to the presentation of one step of the process of learning, shown in fig. 3.6, notice, what goes on further: Having found the direction of the quickest fall of the error the algorithm of learning the network makes the migration in the space of weights, consisting of changing the old (worse) vector of weights to a new one (better). This migration causes that on the surface of the error we „will slide down" to a new point - most often situated lower, that is to say bringing near the network to the longed-for valley in which errors are least, and solution of put assignments - most perfect. Such optimistic scenario of gradual and efficient moving toward the place, where errors are least, is shown in figure 3.7.

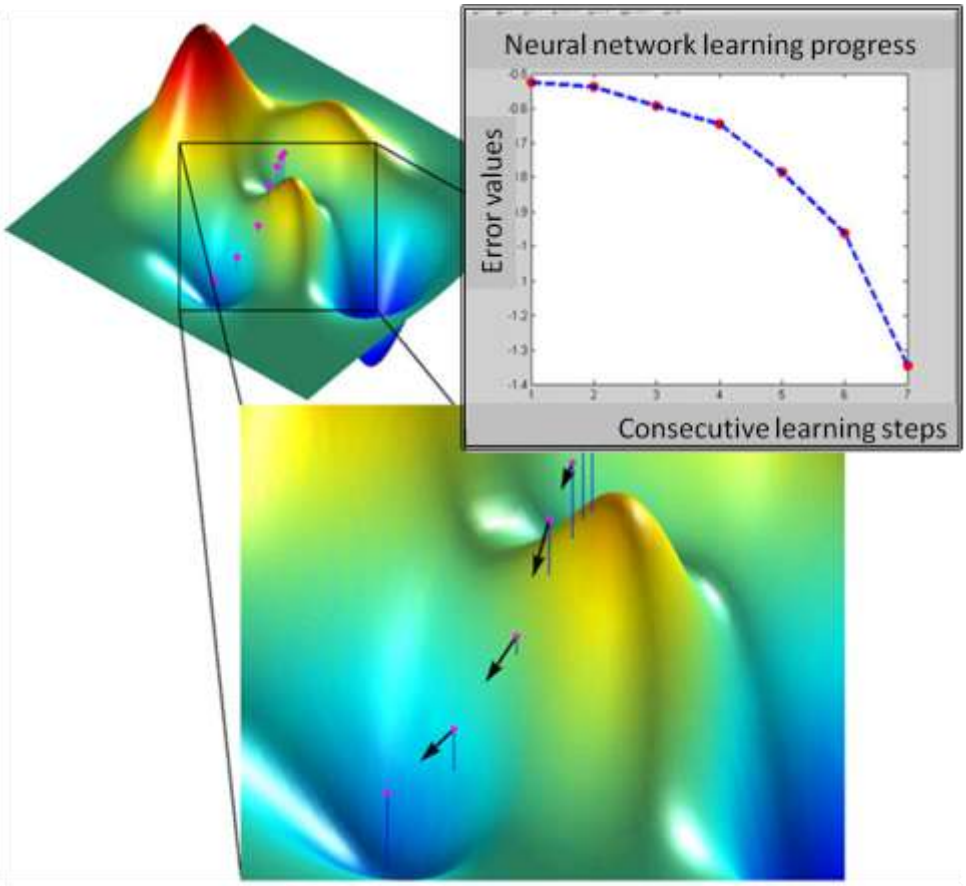


Fig. 3.7. Searching (and fining!) network parameters (weight coefficient for all neurons) guarantying minimal value of the error – during supervised learning.

3.5. Why does this sometimes not succeed?

(Translation by Agata Krawcewicz, hogcia@gmail.com)

Not always is learning such a simple and pleasant process, as I showed in fig. 3.7. At times the network „struggles" for a long time, before it finds a required solution. The detailed discussion of difficulties appearing here would again demand references to complicated considerations on the subject of the infinitesimal and differential calculus together with the analysis of the convergence of algorithms, and with other rather difficult things. However, instead of persecuting you with complicated mathematics I will try to explain it, by telling you a story of a blind kangaroo (look at fig 3.8).

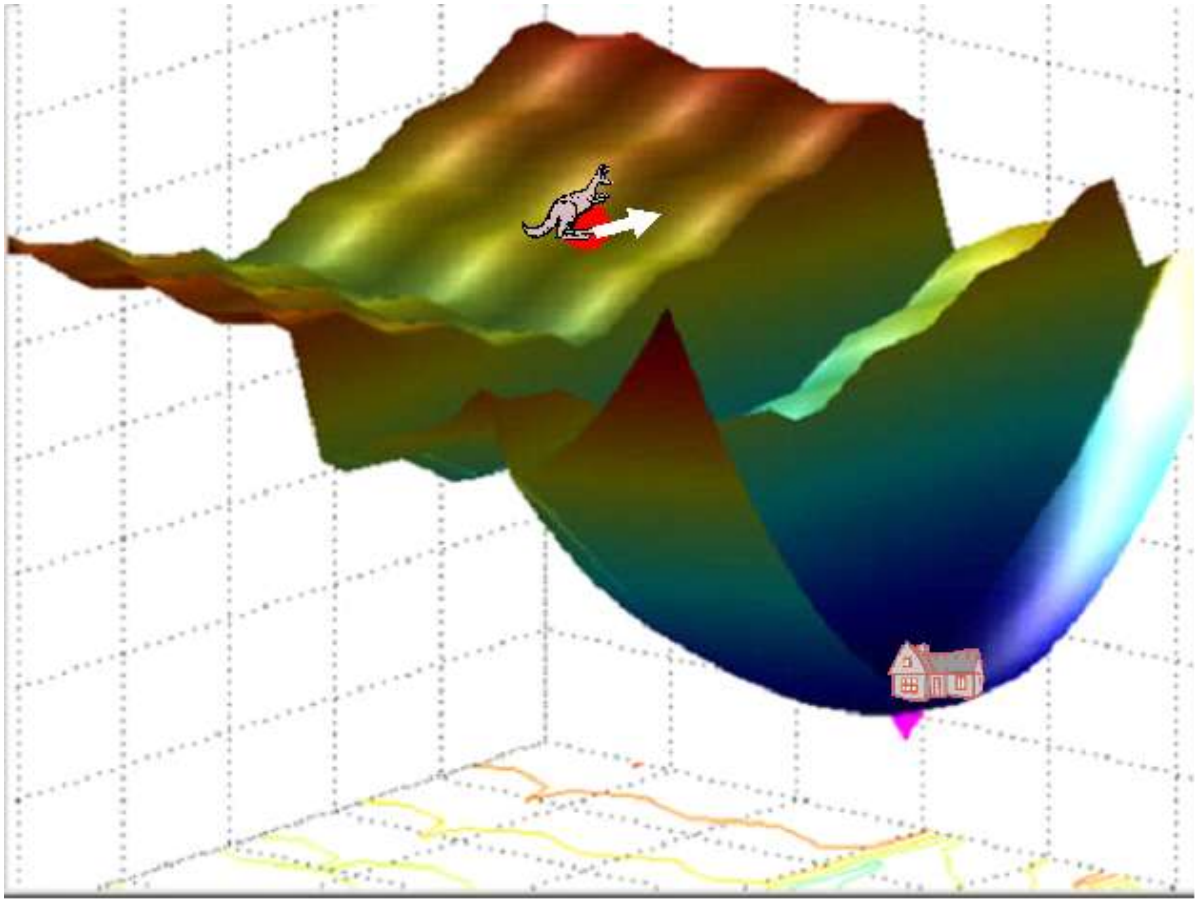


Fig. 3.8. Learning of neural network presented as blind kangaroo hike. Discussion in text.

Imagine that a blind kangaroo got lost in the mountains and wishes to come back to its own house about which it knows only the fact, that it is situated at the very bottom of the deepest valley. The kangaroo thinks of a simple method: He is indeed blind, so he cannot see all the landscape of the mountains surrounding him (just as the algorithm of learning the networks cannot check the value of the error function in all points for any sets of weights), but can feel with his little paws, which way does the ground subside (in the same way as the algorithm of learning the network can find out, which way it is necessary to change weights, so that the error will grow smaller). So that way the kangaroo finds a proper direction and hop! - it jumps with as much powers as he has in his legs, counting on the fact, that he should aim to his own little house.

Unfortunately a few surprises are waiting for the kangaroo (and on the algorithm of learning the neural network). When you will look thoroughly at figure 3.8 you will notice that the imprudent jump can lead the kangaroo down of the rift which separates him from his house. A situation is also possible (not drawn, but easy to imagine), that the ground can drop in the certain direction - but a little further it can suddenly raise, what will cause that performing a long jump in the seemingly promising direction - the kangaroo will make his situation worse, because he will find himself landing higher (that is to say further from the aim), than he was in the starting moment!

The success of the poor little kangaroo depends mostly on the fact, whether he can properly measure the length of the jump. If it will perform small jumps, then the way home will take him a lot of time. But if he decides on a jump that is too long, and in the environment there are some crags or rifts, he will harm himself!

At learning the networks the creator of the algorithm must also decide, how big the changes of the weight should be, caused by particular values of input signals and the specific size of the error. The decision this is made by changing the so called proportion coefficient - learning rate. Apparently it can be chosen just as we wish, however every particular decision has specific consequences. Choosing a coefficient **that is too small** makes the process of learning very slow (weights are improved very slightly in every step, so in order for them to reach desirable values we have to perform lots of such steps). And choosing **a too large** coefficient of learning causes very abrupt changes of parameters of the network, which in the extreme cases can even lead to instability of the process of the learning (the network tosses, not being able to find the correct values of weights, changes of which being made so quickly, that precise "shooting itself" into necessary solution is very hard).

One can have a look on this problem from yet another point of view. Large values of the coefficient of learning resemble an attitude of a teacher, who is very strict and difficult to please and who too radically and severely punishes the pupil for his mistakes. Such teacher seldom attains good results of learning, because he sets pupils into confusion and causes their excessive stress. On the other hand little values of this coefficient of learning resemble a teacher who is excessively tolerant and whose pupils make too slow progresses, because he insufficiently rushes them to work.

When learning the networks and learning pupils it is necessary to make a compromise, taking into account both advantages related with quick work, and safety considerations, pointing out the necessity of obtaining a stable functioning of the process of learning. You can however help yourself in another smart manner which I will describe to you in the following subchapter.

3.6. What is momentum used for?

(Translation by Agata Krawcewicz, hogcia@gmail.com)

One way of increasing the learning speed without interfering with the stability is the use of an additional component, the so called momentum, in the algorithm of learning. Graphically one can say that momentum enlarges the inertia of the process of learning - changes of the weights then depend on both - the errors currently made by the network, and the course on the process of learning at the earlier stage.

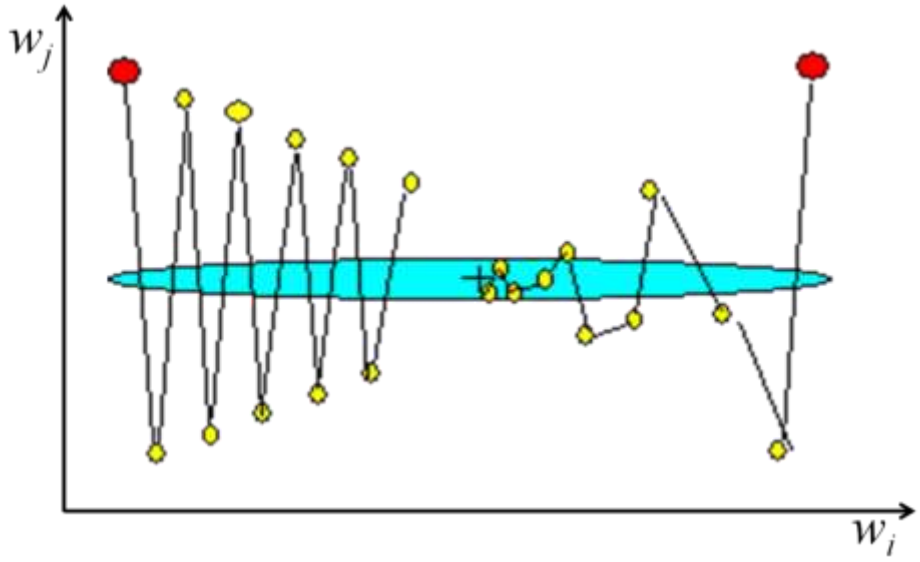


Fig. 3.9. Learning process without *momentum* (left side) and with *momentum* (right side)

The figure 3.9 allows you to compare the process of the learning with momentum and without it. In this figure I showed you the process of changing the weights coefficients. I can show only two of them therefore drawing should be interpreted as a projection on plane determined by weight coefficients w_i and w_j the weight adaptation process, which takes place in the *n*-dimensional space of the weights. In can see only behavior of two inputs for the certain neuron of the network. But in other inputs in other neurons processes are similar.

Red points represent starting points (resulting from the setting - before start the process of learning - values of weight coefficients), and yellow points are the value of weight coefficients obtained in consecutive steps of the process of learning. An assumption has been made that the minimum of the error function is attained in the point ”+”, and the blue ellipse shows the outline of the stable error (set of values of weight coefficients for which the process of learning attains the same level of error). As it is visible in the figure, introducing the momentum really causes the process of the learning to become calmer (values of the weights coefficients do not change as violently and as often), and other than that - more efficient (the consecutive points approach to the point ”+” faster than before, the "+" point being a solution of the problem). Now during the learning the network **from a rule** one uses momentum, because it improves the process of reaching the to correct solutions, and at the same moment the execution costs are not too high.

Other manner of improving the process of learning can rely on the usage of changing values of coefficients of the learning - small at the beginning of the process of learning, when the network only chooses the directions of its activity, greater in the centre-piece of the learning, when it is necessary to act forcefully yet roughly enough to adapt the values of parameters of the network to established rules of its activity, and at last again smaller in the end of the process of learning, at the moment when the network perfects the final values of its parameters (fine *tuning)* and too impetuous corrections can destroy the construction of the earlier built structure of knowledge. Let us notice that these techniques of the activity, about the mathematically led out structure and the experientially examined usefulness lively resemble the elaborate methods of the teacher, who has a large didactic practice and used for pupils with a small psychical strength! Is without a doubt the

striking convergence of the behavior of the neural network and the human mind - not first, and not last after all.

3.7. Where should we start when learning the network?

(Translation by Agata Krawcewicz, hogcia@gmail.com)

Every creator of an algorithm of learning of neural network has to bare with the problem, of original values of coefficients of weights. The algorithm of learning described earlier shows, how one can **improve** values of these coefficients in progress of the process of learning, however, we have to start this process somewhere, because already in the moment of showing the first object of the learning he network must have some concrete, settled values of coefficients of the weights, to be able to qualify values of output signals of all neurons, and then to compare it with values given by the teacher for the purpose of the qualification of errors. Still, where should we get these original values of all weights from?

The theory of the learning neural networks says that at the realization of certain (simple enough) conditions the lineal network can learn and find correct final values of weights - (resolving this problem - if simply such solution exists) coefficients **not depending on** which initial values we will take at the start of the process of learning. In the domain of non-linear networks the situation is not so simple, because the process of learning can get stuck in the starting values in the so called minima of local error functions, what in effect means that starting from different primary points (from different primary sets of weight coefficients) we can receive **different** parameters of the taught network - better or worse fitted for resolving the set assignment. So does the theory say, but does it say anything about how to choose these good, advantageous starting points?

Here we have to, unfortunately, support our knowledge on empiricism. A simple and easily used practice relies on fact, that the primary values of weight- coefficients are **random.** At the beginning all placed parameters receive accidental values, so we start from some not earlier presumed random values. It may at the first moment sound strange, but this makes sense - if nothing it is known about where to seek the necessary minimum the error function, there is no better solution as to rely on the blind fate!

In simulator programs there is a special instruction "sowing" the first random values of coefficients, in the network; whereat we should avoid too large values of start coefficients (regularly practiced is the section from -0,1 to +0,1, whence they are chosen randomly). Additionally in many-layered networks one also has to avoid coefficients having value 0, because it blocks the process of learning of deeper situated layers (through a zero value of weights signals do not pass, what in practice means „cutting off ” a part of the structure of the network from further learning).

Of course the progress of the process of learning reduces the error. This process is very fast at the beginning, later however becomes slower, and at last after realization of a number of steps - the process of learning stops completely. It is useful to remember that the process of the learning targeting training of the exercise of the same assignment by means of the same network - can run differently. In fig. 3.10. I showed you three courses of this process which differ both with the speed of the progress of the learning, as and with the final effect (in the form of the size of the coefficient

of the error, below which the network no longer can to go down, in spite of the intensity of the further learning). Because all graphs refer to the same network and the same assignment, and differ only with primary values of coefficients of weights - we could say that these graphs show diverse „inborn abilities" of the networks.

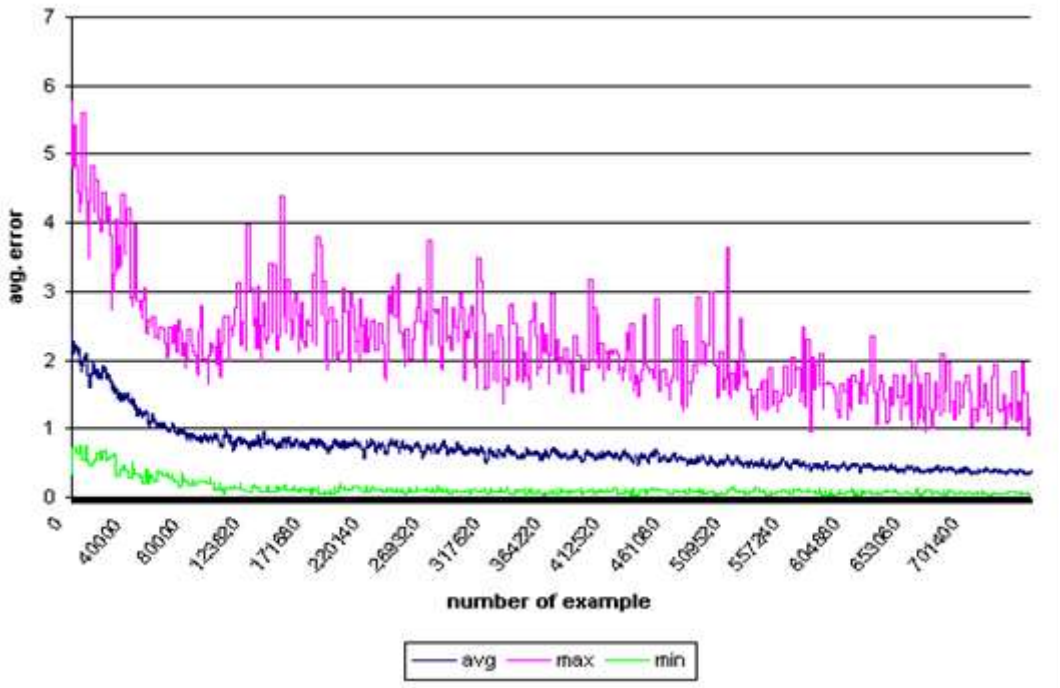


Fig. 3.10. Decreasing of network error during the learning process. The same neural network learned many times using the same learning set, but starting from different points (determined by random process of weight initial values setting). Three courses are plotted: average (most representative), course with maximal end value of the post-learning error (pessimistic case) and course with minimal end value of the post-learning error (optimistic case).

In some types of network (for example Kohonen network*)* one demands additionally, that original values of weight- coefficients to be somehow normalized, but I will talk about this during the discussion of these particular types of networks.

3.8. Is learning a network a long process?

(Translation by Agata Krawcewicz, hogcia@gmail.com)

Unfortunately, the answer to the above question is pessimistic. For the comfort of using the process of the learning instead of "the manual" programming of the network, a price must be paid with a long period of the learning - and in general it is necessary to humbly accept this fact, because no one has yet found a really efficient solution to this problem. Worse still, it is difficult to anticipate, how long it will be necessary to teach **the concrete, given** network, before it will begin to develop some elements of an intelligent behavior. Sometimes it is going like a lightning, but in general it is necessary to show elements of the learning set for a long time and with great effort, before the network will begin to understand at least a little of what we demand from it. What is more, as the

result of random choice of original values of weight- coefficients the results of learning **of the same** network for the same data used as the learning set can differ a little and this fact should be also simply accepted - unless somebody has time to teach the same network several times and afterwards to choose the variant which guarantees the best result. However, in practice this is very time-consuming, because in my research and the research of my candidates for a doctor's degree I have found many times that a single process of learning a network with several thousands of elements lasts **several dozens of hours!**

This is not an isolated result. For example from the compilation published in the monthly *Electronic Design* we can see that the number of the presentations of the learning set, necessary to achieve the correct work of the network in chosen uses brings out:

- for the character recognition of Kanji - 10^{13}
- for the speech recognition - 10^{12}
- for the diagnostics of manuscripts - 10^{12}
- for the voice synthesis - 10^{10}
- for financial prognosis - 10^{9}

Other authors bring similar results.

Because of the speed of learning people seek different (from classical, described here, delta algorithm) new methods of learning the networks. It is considered that the time of learning the networks with the algorithm of the quickest fall (described earlier) grows exponentially with the advance of the number of elements of the network what can in the future be one of basic factors of limiting sizes of built networks.

3.9. How to teach hidden layers?

(Translation by Agata Krawcewicz, hogcia@gmail.com)

Lastly we will discuss one more problem - the problem of learning networks, connected with the presence of hidden layers. As I described in the previous chapter what we call a hidden layer is every layer except the entrance layer and exit layer. The essence of the problem appearing at the learning of neurons of the hidden layer consists of the fact that for these neurons we cannot directly set the size of errors, because the teacher gives standard-values of signals only for the exit-layer , and for the signals appointed by neurons of the hidden layer (probably also incorrect at the beginning of learning) we have nothing to compare them with. Certain solution for this problem is the method of the so called backward propagation of the error (backpropagation*)*, for the first time suggested by *Paul Werbos,* and afterwards popularized by *David Rumelhart* and bound usually with his name.

This method consists of reproducing the presumable values of errors of deeper (hidden) layers of network on the ground of projecting backwards the errors detected in the exit-layer. Roughly one can say (at this moment - because in further chapters of the book the process will be put to a

detailed analysis) that we do it like this: considering every following neuron of the hidden layer one takes under caution the errors of all the neurons to which it sent its own output signal and **one adds it up ,** taking into account the values of coefficients of weights of connections between considered neuron and neurons whose errors are added up. This way "the responsibility" for errors of neurons of the hidden layer burdens the neuron of the hidden layer the more strongly, the more heavily (because with the greater weight) it influenced the signal of the given hidden neuron for elaboration of definite signals of the exit-layer (that is to say at the creation of definite errors detected at the stage of evaluation of the network's answer).

Proceeding consistently in this manner and moving step by step from the exit to the entry of the network we can evaluate presumable errors of all neurons, and consequently obtain premises permitting us to qualify the necessary corrections of weight coefficients of these neurons. Of course the described proceedings does not ever permit to mark errors of neurons of intermediate layers exactly, so corrections of errors in progress of the process of learning are less accurate and less exact, than in the case of networks without hidden layers. This effect appears in practice as a considerably longer time of learning such networks, which I wrote earlier. However, taking into account the considerably wider range of many-layered network's capabilities we have to accept the fact, that it is proper to carry the cost of larger complexity of the process of learning, even if the necessary result (in the form of the correctly trained network) will we succeed to reach only after tens thousands of demonstrations of all objects of the learning set.

With a repeated presenting of all objects of the learning set in the progress of learning the neural network certain additional problems come up. The necessity of such presenting of learning set "over and over again" comes out of the fact that even at the presumption of an angelic patience of the teacher we can accumulate as a material to learning the networks tens, sometimes several hundred, and in sporadic cases several thousands of examples - however it is never hundreds of thousands or millions, and so many steps of learning are often necessary to teach the networks. So there is no other way out - it is necessary to present the same objects (this means input signals and patterns of the answer of the network) many times. It is definitely disadvantageous to show the learning set constantly in the same order of elements. Such periodical instruction can lead to appearing of definite cycles of the value of weight- coefficients , what does not forward the process of learning and was the cause of many, well-known to me, failures. So it is very important to randomize the process of the learning - the accidental mixing of the learning set and showing its elements to the network in a different order every time. It complicates the process of learning a little, but in most cases it is a condition of obtaining reasonable results.

3.10. How can a network learn by itself?

(Translation by Agata Krawcewicz, hogcia@gmail.com)

The question put in this subsection's title directs us to problems known in the literature under a little perfidious name of "learning without a teacher". The idea of the task which has to be solved leads to such a situation: a network should by itself perfect the manner of resolving a particular assignment without having a set of ready examples with solutions. Still, is this at all possible? How can we expect that the network will perfect its own work, not giving it any clues for this task?

Even though it proves to be entirely possible and even relatively easily attainable to have a spontaneous self-organizing neural network - with considering several simple conditions. The simplest idea of self-learning of a network is based on the observation of an American physiologist and psychologist , Donald Hebb, that in the brain of animals (and in the mind of a man) happen the processes of strengthening (amplification) of connections between neurons (or with all sets of neurons, the so called centers), under a condition that they were stimulated to work **simultaneously.** In such a way the mind associations are created, and (by what Hebb states) reflexes are shaped, in such a way also simple forms of motor and perceptive skills come into being.

Transferring the Hebba theories on the ground of neural networks computer scientists designed a method of self-learning which consists of showing the network examples of input signals, **without giving any information on what should it do with them.** The network simply observes the environment and takes different signals, even though no one tells it, what meaning do showed objects have and what dependencies exist between them. The network, on the ground of observing the occurring signals gradually discovers, what is its meaning and also sets the dependencies existing among signals. This way the network not only acquires knowledge - it in a way **creates it!** Let us now trace this interesting process a little more thoroughly.

After passing every consecutive set of input signals, the network forms a certain distribution of output signals - some neurons of the network are stimulated very intensely, other weakly, and some yet another have output signals which are even negative. The interpretation of these behaviors can be that some neurons "will recognize" given signals as " its own" (that is those which are likely to be accepted), other treat it "indifferently", while other neurons are simply "disgusted". After the settlement of output signals of all neurons in the whole network - all weights of all neurons are changed, whereat the size of the suitable change is calculated based on **the product of the input signal,** entering the given entry (which weight is being changed) **and the output signal** produced by the neuron, whose weights we modify. It is easy to notice that this is just the realization of the Hebb postulate: in the effect of the algorithm described earlier connections between sources of strong signals and with neurons which react strongly on it - are strengthened.

More exact analysis of the process of the self-learning with the Hebb method allows us to state that a result of the consistent using of described algorithm the start parameters, most often accidental "preferences" of neurons submit to the systematical strengthening and detailed polarization. If some neuron had "an inborn inclination" to accepting signals of a certain kind - then during following demonstrations it will learn to recognize these signals more and more accurately and precisely. After a long time of such self-learning the network will create (quite spontaneously!) **patterns** of each **types** of signals given on the network's entry. As a result of this process similar signals will be recognized in the progress of learning more and more efficiently **assembled and** recognized by certain neurons, while other types of signals will become "an object of the interest" of other neurons. As a result of this process of self-learning the network - all by itself, only by observing given signals - will learn , how many classes of similar signals appear on its entries and assigns the neurons which will learn to differentiate, recognize and signal these classes. So little and as much**!**

3.11. How should we conduct self-learning?

(Translation by Agata Krawcewicz, hogcia@gmail.com)

The process of the self-learning (or maybe self-learning) described earlier has, unfortunately, faults. Compared to a process **of learning with a teacher,** the such self-studying is usually considerably slower, so if there is a possibility of choice - one ought to choose controlled learning, rather than spontaneous. What is more, without a teacher we never know from the beginning **which** neuron will specialize in the diagnostics of which class of signals (for example the first can stubbornly recognize the letter R, second D, third G - and no force can make these neurons arrange itself in an alphabetical order). What this determines is a certain difficulty at using and the interpretation of results of the work of the network - for example in the system of steering of a robot. The indicated difficulty has yet a more basic character and more troublesome results - there is namely no guarantee that developing its own starting, random preferences, neurons will specialize to the extent that every one of them will show **different** class of entrance-images. On the contrary, it is extremely probable that several neurons will insist to recognize the same class of signals - for example several neurons will recognize the letter A, while none "will decide" to recognize the letter B. That is why the network intended to do the self-learning must be **greater,** than a network solving the same assignment, but trained in a classical way, with the participation of a teacher. It is difficult to measure this exactly, but from my own experiences and from the observation of students writing their master's theses in my laboratory there seems to arise a result that the network intended to be self-learning must have at least **three times** more elements (especially in the exit-layer), than it would get out of the number of the answers, which the network should state after learning.

A very subtle and essential matter is the choice of the start values of weights of neurons of the network intended to be self-taught. These values have a very strong influence on the final behavior of the network, because the process of learning only deepens on and perfects certain tendencies existing in the network from the very beginning, therefore from these first, "inborn" proprieties of the network, it strongly depends what the network will reach by the end of the process of learning. Not knowing from the beginning, which assignment the network should learn, it is difficult to introduce any determined mechanism of setting the start values of weights, however leaving it all only to random mechanisms can cause that the network (especially small) may not manage to sufficiently diversify its own mechanisms in the first period of the process of the learning and all later efforts to find the representation for all occurring entrance-signals of classes in the structure of the network, can prove to be futile. However, we can introduce one mechanism of initial "distributing" of the value of the weights, in the first phase of the process of learning. The method, called convex combination *(*described more exactly in my book "Neural networks", accessible in the Internet under the address ***http://winntbg.bg.agh.edu.pl/skrypty/0001/)*** modifies original values of the weights in a way, that is intended to maximize the probability of an even distributing of all typical situations appearing in the entrance-data set, by particular neurons . If only data appearing in the first phase of the learning will not actually differ from those, which the network will afterwards analyze and differentiate - then the method of convex combination will automatically create a convenient starting point to the further self-learning and will assure the comparatively good quality of the taught network in most of practical assignments.

3.12. Supervisory questions and problems to solve.

(Translation by Agata Krawcewicz, hogcia@gmail.com)

1. What conditions must be fulfilled by the data set, so that it can be used as a teaching set for the a neural network?

2. How does teaching the network "with a teacher" differ from teaching „without a teacher"?

3. On what grounds does the algorithm of teaching calculate the direction and the amount of the change of weights in the steps of the process of teaching?

4. How should we set the starting values of weight coefficients of a network, to start it's process of learning?

5. When does the process of teaching stop by itself?

6. What are reasons of failures in the process of teaching the network and how can we prevent them?

7. What does the process of generalizing the knowledge obtained by the neural network in the process of teaching consist of?

8. Where can we obtain the data necessary to fix, in the process of teaching, the value of changes of weight coefficients of the neurons which are in the hidden layers?

9. What is momentum and what can we do with its help?

10. How long a process of teaching do big neural networks require? What does it depend on?

11. Why, in the case self-learning networks, do we frequently speak about "discovering knowledge", and not simply about teaching the network?

12. **A more difficult assignment, for the advanced:** Try to suggest a rule, in compliance with which one can change the coefficient of the teaching (learning rate*)* in progress of the process of teaching in such manner, that it would accelerate the speed of learning in safe situations, and slow it down at the moment, when the stability of the teaching process is endangered.

An advice: Consider changes of the value of errors committed by the network in consequent steps of the process of teaching.

4. Functioning of a simplest network

4.1. From theory to practice – how to use programs dedicated to the readers of this book?

(Translated by Krzysztof Kajdański; krzysiek.kajdanski@wp.pl)

The main purpose of the three previous chapters was to provide theoretical basis concerning neural networks. Starting with this chapter you will begin to gain practice in using them. From this moment I will do my best to illustrate all the further argument with the help of the simple programs, which will allow you to try on your own how the neural networks work and what the basic rules of their usage are. These programs (as well as the data needed for their start-up) can be found at:

http://www.agh.edu.pl/tad

When you input this address into your web browser, you will see a site, which describe accurately and step by step, all actions you need to take to get (free of cost but totally legal) these programs, which we are going to use together further in this book. You need these programs not to believe uncritically in what I am going to write about the neural networks. Having these programs installed on your computer you will be able to discover different features of the neural networks, carrying out very interesting experiments and having a lot of fun with them. Don't worry – all you have to do is to run one or two typical installation programs. If you ever tried to install anything on your computer, you would be able to do it. The detailed information about the installation is at the website mentioned previously. As the software use to be updated, the detailed description in this book would quickly become outdated. However I will allow myself to bring up a few boring (alas!) details concerning what you should do.

Downloading the programs from the website, which address I previously mentioned, is very easy and can be done with one mouse click (in case you need help ask someone who have downloaded a file from the Internet before). However getting only the programs is not enough. There are written in C# language and need installed libraries, which are called the **.Net Framework** (2.0 version). If you have no idea what is this all about – don't worry - you will be able to find all needed software at the website.

The first step you should take is the installation of the libraries mentioned. Of course, it may occur that you have these libraries installed on your computer already and this step is unnecessary. If you are sure about that, you can skip the **.Net Framework** part of installation, but I advise you to perform it, just in case. If the appropriate programs are installed on your computer, installer will find out on its own, that there is no need to install them. However it may happen, that at the site mentioned there are newer versions of the libraries than the ones you have. Then the installer will get down to work, which should be ok, because it is always good to exchange old software for more modern. This new software will become useful in many purposes, not only the one concerning studying this book and performing described in it experiments.

Very important notice

All software available on site

http://www.agh.edu.pl/tad

is legal and free of cost on terms of the educational license of **Microsoft®**.

This license allows free use and development of the **.NET** technology on condition it is not used for commercial purposes. It means that you can use this software for experiments described in the book with no limits. You can also use it for your own programs – as long as you don't sell them.

Let's go back to the installation of software required.

When you finish the installation of the necessary **.Net Framework** libraries - the necessary part of all your further actions - install the example programs according to the guidelines contained at the WWW site mentioned. The installer will run again, which allow you automatically (and from your point of view – totally painlessly) install all written by me example programs. You are going to use them during further studying of this book. Thank to them you will be able to get to know the neural networks theoretically, and, what's more, play with them in a very practical way (being more serious: perform necessary experiments :)) and by the way get very useful practice in the application of the neuron calculations mechanism. When installer's work is finished, you will be able to access them through the **Start** menu, in the same way as the majority of your programs.

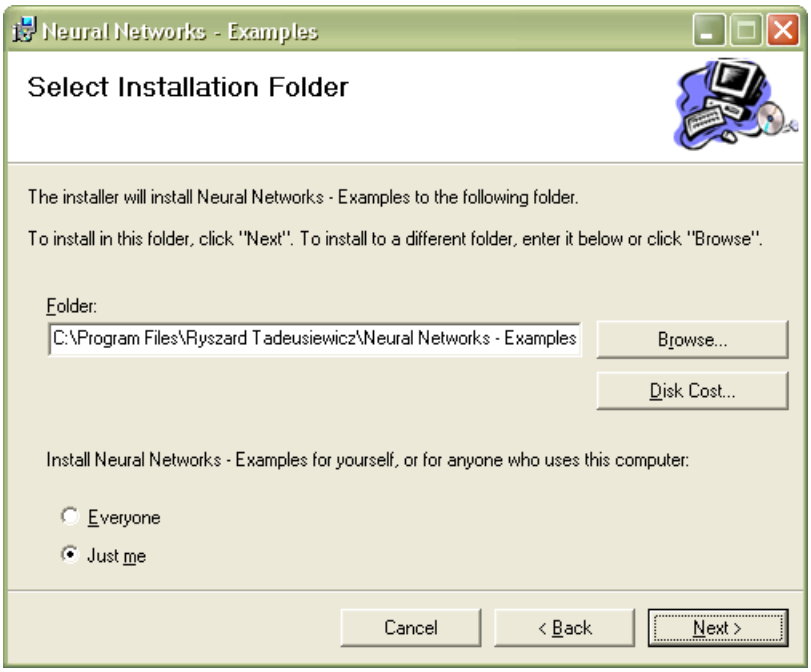


Fig. 4.1 The installer is going to ask in fact only two questions

To convince all the unbelievers that this is simple, in the picture 4.1 there is shown the only hard part of installation, in which installer ask the user, where to install the programs and for whom there should be accessible. The wisest thing to do is to leave default values and just click **Next**.

For more inquisitive adepts of computing science (you are one of them, don't you?) there are two others options: downloading the source code and the installation of the integrated development environment **Visual Studio .NET**.

The first option allows you to download to the hard drive of your computer human readable (it means understandable to you!) texts of all of these sample programs which executable versions you are going to use during studying this book. It is very interesting to check, how all of this is made, and why every program works as it works. Having the source code will allow you – if you dare to – to modify the programs and perform even more interesting experiments.

Although I would like to emphasize that the source codes are not necessary if you just want to use the programs described in the book, but if you have code, you will be able to learn more and use the resources I prepared for you in the more interesting way. It is tempting especially that obtaining the codes is not laborious.

The last option – **Installation of the Integrated Development Environment Visual Studio .Net** is addressed to those of you, who would like to modify our programs, to perform your own experiments, or basing on their components – write your own programs. This is option for ambitious readers. However I encourage you to do that even if you just want to look through the code. It is worth to spent a few more minutes to make viewing of the code much more easier. **Visual Studio .Net** is very easy to use *Integrated Development Environment* adjusted to convenient edition and browsing of the code. Having **Visual Studio .Net** installed you will be able to perform many more additional actions such as including extern sets, easy diagnosis of applications and quick generation of complicated ultimate versions of the software.

Remember, that the installation of both the source code and the **Visual Studio .Net** is fully voluntary and you can use it or not. To run the example programs, which will allow you to create and experiment with described in this book neural networks, you only need to install **.Net Framework** (first step) and example programs (second step).

All right, you copied and installed everything what was needed – what's next?

Now to run any example you need to choose appropriate command from **Start/Programs/Neural Networks – Examples** menu. After choosing the appropriate program you may use your computer to create and analise every neural network described in the book. Initially it will be a network with the shape and measurments designed by me, but when you immerse yourself in the source code, you will be able to modify and change everything you want. Started program will make network live on your computer, and it will be able to be teached, tested, analised – summarising, examined. I think that this way of **discovering** the features of the neural networks – through construction and making them work – may be found more inspiring and imagination arousing than any other lecture or theoretical reasoning.

The way the network works is dependent on its structure and purpose. That is why we are going to consider in the further chapters a few specific cases. Because it is the simplest to explain the work of the network that **recognizes images** (besides – I know a lot about image recognition – about other applications – not necessarily) and that is why we will start our discussion with it.

This kind of network – as mentioned before – gets an image as an imput, and as an output signals, to which of previously learned classes object seen in the image belongs. This kind of network was presented in the picture 2.34 which you have been watched lately. The recognizing tasks are fulfilled by the networks classifing geometrical figures, identifying printed and hand-writen letters, either planes silhouttes, or people's faces.

How this kind of network works?

To answer that question let us start from the absolutely simplified network – the network containing only one neuron.

- What? You say it is not a network, because in the network there should be a lot of neurons and they should be **connected** to each other?

- Well, it doesn't matter. It will turn out that even so **small network** can act quite interesting!

4.2. What can be expected from a single neuron?

(Translated by Krzysztof Kajdański; krzysiek.kajdanski@wp.pl)

As you probably remember – neuron gets input signals, multiplies them with factors named weights (for every input set individually during the learning process) and then sums created in that way ratio. Let us stop for a while on this simplified model. Although you know already that in more complicated neurons these summed signal are converged to the output signal with an appropriate function (generally non linear one), but believe me - in our first simplified neuron it is not necessary. We will examine the behavior of this simple linear neuron. What does the value of the output signal depends on?

It is easy to show that it depends **on level of acceptance between the input signals on every input and the values of the weights on these inputs**. Although this rule matches ideally only for the **normalized** input signals and weights (I'll specifically explain later how you should understand it). However even without specific normalization the value of the output signal can be treated as the measure of the similarity between assembly of the input signals and the assembly of the corresponding weights.

You can say that a neuron has its own memory and stores there the representation of its knowledge - the pattern of the input data, which it is sensitive for (in the form of the actual values of the weights). If the input signals match the remembered pattern – the neuron 'recognizes' them as something familiar and answers to them with the strong output signal. If there is no connection between the input signals and the pattern – output signal in near to zero (no recognition). There is possible a total contradiction between the input signals and the weights values. The linear neuron generates the

negative output signal then – the stronger contradiction between the neuron's image of the output signal and its real value, the stronger output.

To begin with I encourage you to try to run the simple program named **Example 01a**. You can start it and just perform a few experiments that program allows on. You can benefit even more if you decide to change this program or even try to make it better. Then you will learn much more about the network. After initialization of the **Example 01a** program you will see the window shown on Fig. 4.2. In its top part there is the text explaining what actually we are going to do.

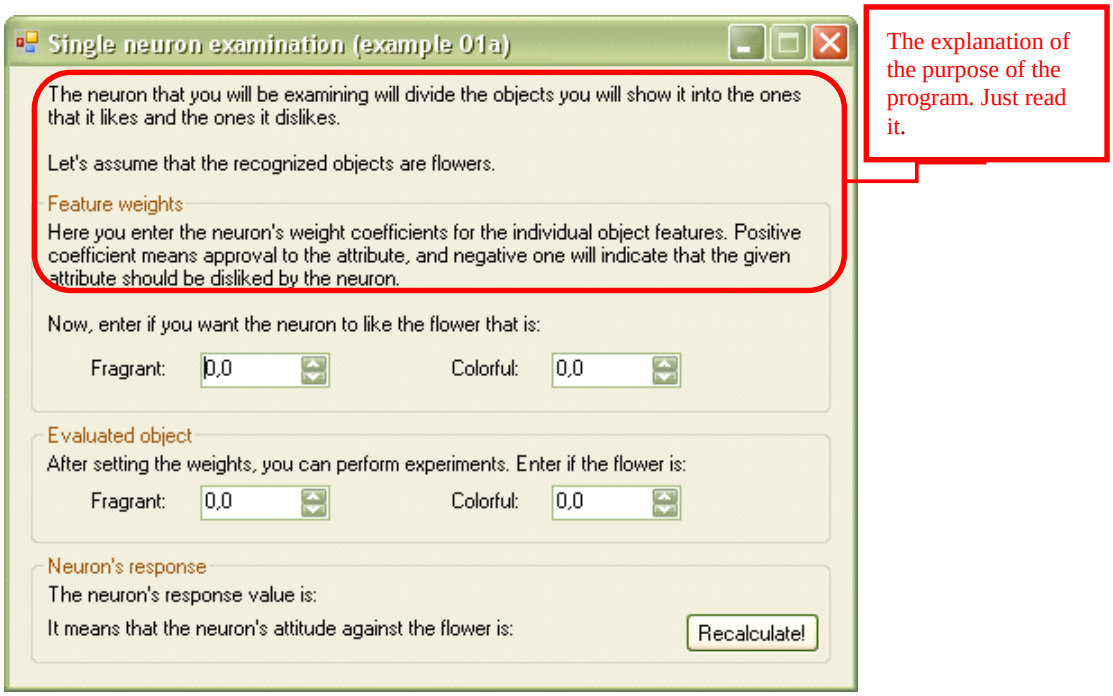


Fig. 4.2. The window of the **Example 01a** program just after start

Blinking cursor signals that programs is waiting – for you! - to give it the weight of the neuron's input which is connected to the **fragrant** value. You can enter it manually by typing a number, or by clicking the arrows next to the field, or by pressing the up and down arrows on the keyboard. After inserting the value for the fragrant feature, go on to the next field, which corresponds to the flower's second feature – **color**.

Let us assume that you want your neuron to like colorful and fragrant flowers, with more favor for color. After giving an appropriate answer the window can look like in the Fig. 4.3.

Described program, as every other you will use, gives you the possibility to change your decision and choose another input data. Program is trying to update the results of its calculations.

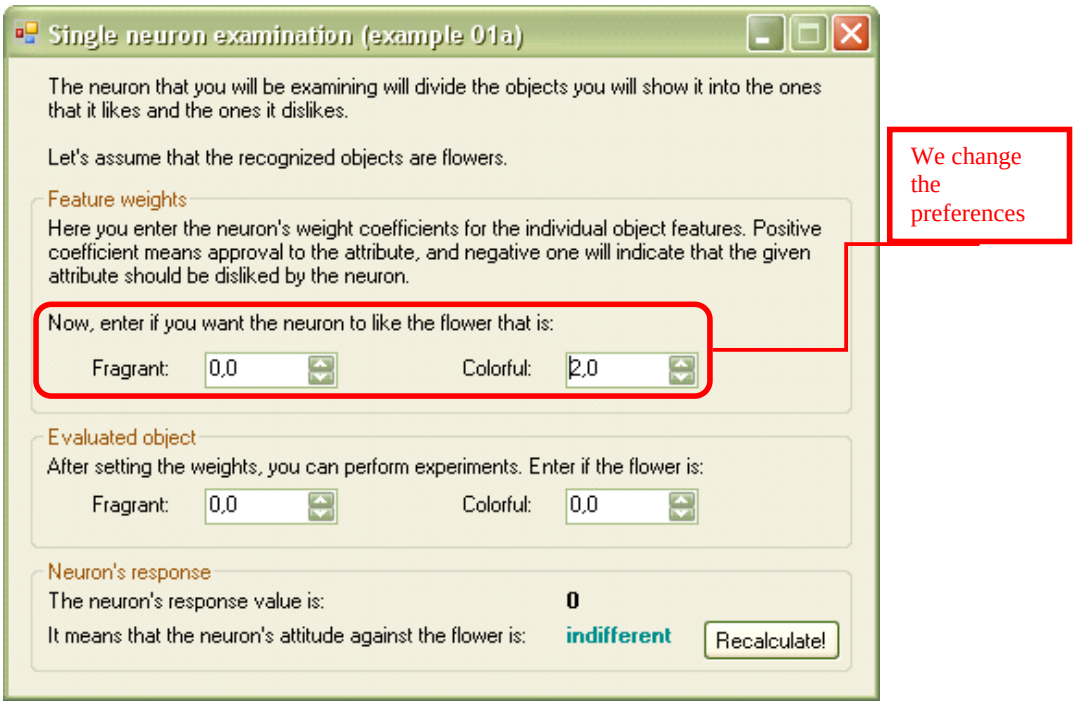


Fig. 4.3. The initial stage of the user's communication with the Example 01a program.

After inputting the data concerning the neuron's features (in fact these are values of its weights) there is time to check the way neuron works. You can input different sets of data in a way shown in the 4.4 Fig. and then the program will calculate and answer what output signal the neuron has generated and what it means. Remember that you can change the neuron's preferences and the description of the flower at any time.

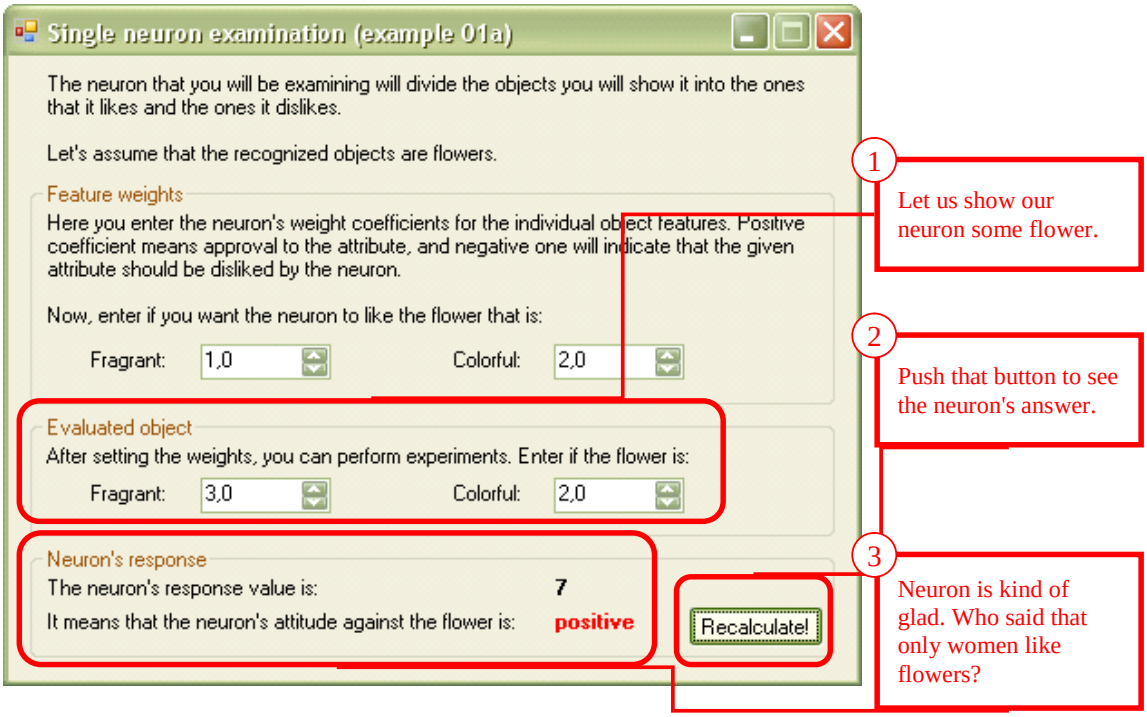


Fig. 4.4. The final stage of the user's communication with the **Example 01a** program.

Remark for the inquisitive readers: If you use mouse or arrow keys to input data you don't have to – as you have probably noticed – click the **Racalculate** button every time you want to see the result; calculations are made automatically. When you input the number from keyboard you have to click it. That is because computer doesn't know if you have finished entering the number or you are just in the middle and went to get yourself a tuna sandwich.

The next stage is to experience our neuron with unusual situation. The point of the experiment shown in the Fig. 4.5 is to observe how the neuron reacts for object that differs from remembered 'ideal' (colourful and fragrant flower). We showed it a flower full of colors but with no fragrance at all. As you can see – it liked this flower as well!

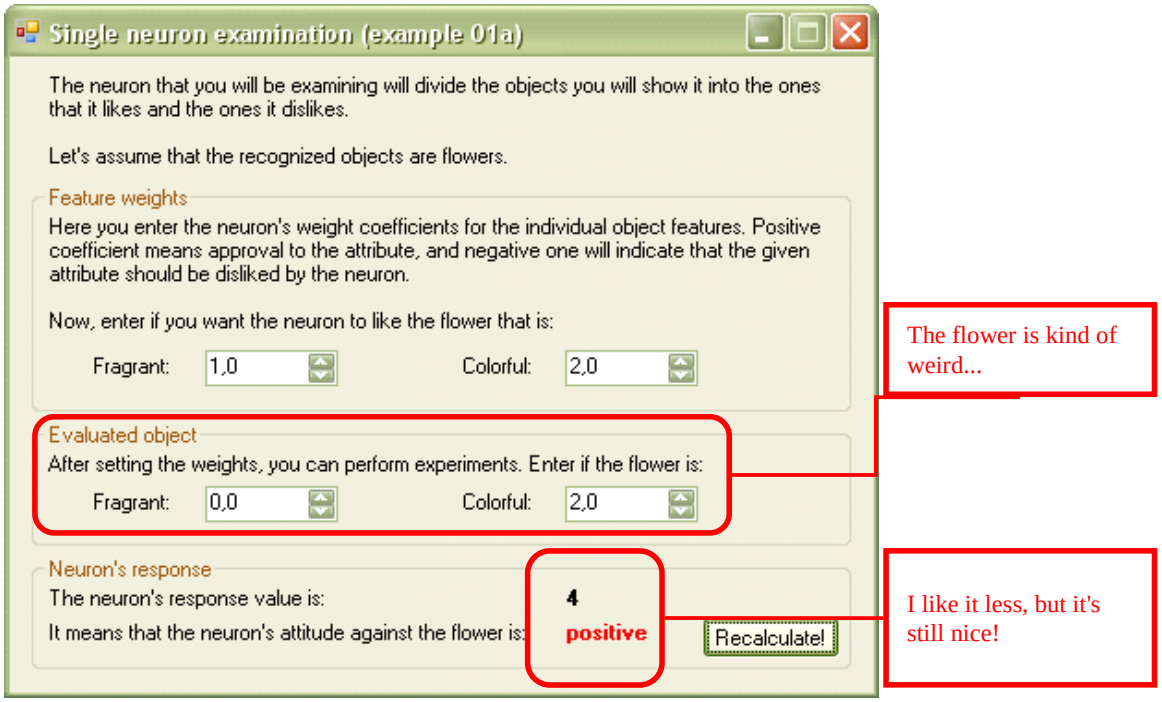


Fig. 4.5. The program **Example 01a** in unusual situation.

Changing the parameters of the flower you can observe what the neuron will do in any other circumstances. The examples of such experiments are shown in the Fig. 4.6. I tested the behavior of the neuron when the flower has nice fragrance, but practically no color (it likes it anyway), and then colorless and smelling badly (this one is not likeable at all).

Fig. 4.6. The example of the another experiment with the **Example 01a** program

And now let us test our neuron with a little more complicated task: let us check if it accepts a flower that smells badly, if it is colorful enough.

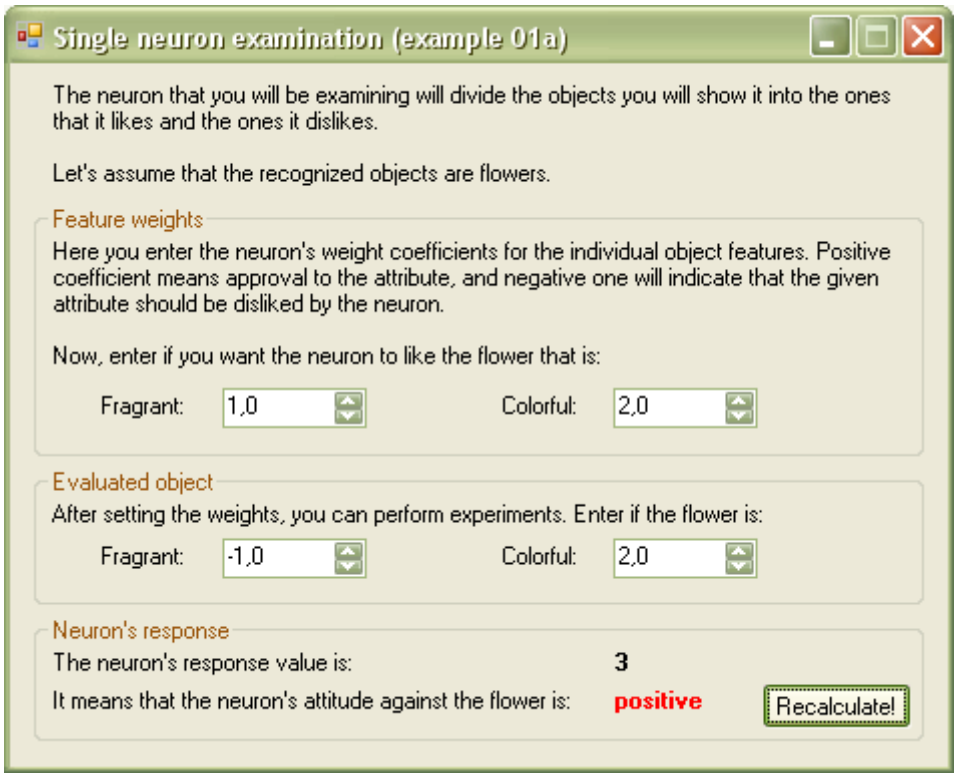


Fig. 4.7. One another example of the experiment with the **Example 01a** program

As you can see, there is a plenty of ways to experiment. You can also try to change the preferences of the neuron and see how it acts in different situations when it, for example, likes when a flower smells bad (the weight corresponding to fragrance can be negative).

4.3. What is worth of noticing during further experiments?

(Translated by Krzysztof Kajdański; krzysiek.kajdanski@wp.pl)

I suggest that you spend some time playing with the program – I assure you it is worth it. While entering different sets of data on input, you will discover fast that the examined neuron works according to the quite simple rule. It just treats its weights as the model for the input signal it wants to recognize. When there is given such a combination of signals that corresponds to the weights of the neuron – it finds in it something familiar and reacts enthusiastically – you receive high output signal then. When different signals are given – the neuron shows indifference (low output signal) or even the aversion (negative output signal). Well, that's its nature!

Careful examination will indicate that the behavior of the neuron depends only on the **angle between the vector of weight and the vector of the input signals**. Have a look at it and use the next program which will also state which flowers neuron likes and which dislikes – this time the model of the ideal flower will be represented as the point (or vector) in so called input space.

The term of the space (and connected with it the weight space) is quite important and quite simple at the same time. It is important for you to focus your attention on this subject for a while – it is not very complicated, though at first time it seems to be the secret knowledge.

When you set the preferences of a neuron – it means when you tell him how much it likes fragrant flowers and independently colorful flowers – you in fact set the parameters of the, so called, **weight vector**. You can draw two axes and on one (let's say – the horizontal) you can mark the values of the first feature (fragrance) and on the other (the vertical for example) the values of the second one (Fig. 4.8). If you want to mark the preferences of the neuron – you can mark a values on the axes and then the point created by these coordinates will correspond to the neuron's preferences. For instance, a neuron that 'values' only the fragrance of a flower and color is rather indifferent to it – will be represented by the point located maximally to the right (high value of the first coordinate) but on the horizontal axis of set (zero or low value of the other coordinate). Apart from that, the neuron that you want to like puttyroots – flowers of beautiful colors and weak, sometimes even unpleasant, smell – will be located high on the top of the vertical axis (high value of color) but even on the left of this axis (acceptance to not nice smell).

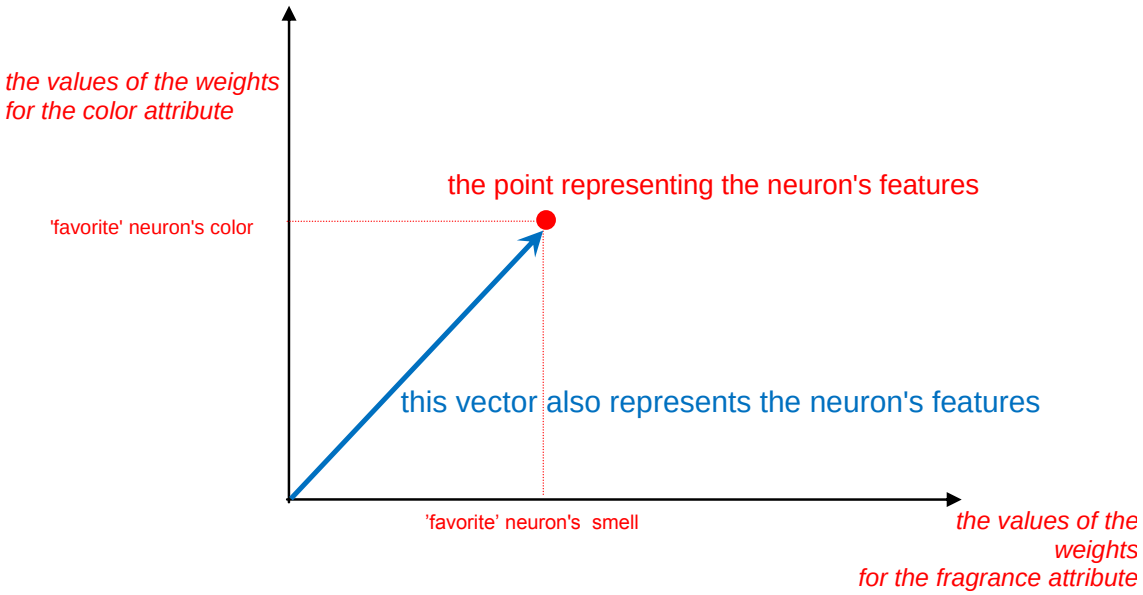


Fig. 4.8. The representation of the neuron's features as the point and the vector in the attribute space

Identically you can treat any other object (flower) that you present the neuron to mark. Its color will be on vertical axis and its fragrance on the horizontal one. If it is Lily-of-the-Valley it will be represented by the point located maximally to the right (it's hard to get anything smelling nicer) but definitely low (color is not the advantage of this flower). If you decide to show it a gillyflower it will be located high (it's generally nice coloured) but definitely to the negative site of the horizontal axis (the smell of gillflower is hard to consider as nice). In the bottom left part of the coordinate system you will find a sundew (it is a plant that consumes insects – it lures flies with the appearance of rotting meat and similar smell), and majestic roses will be found in the top right corner.

During consideration concerning neural networks it is convenient to mark objects (tested flower and the images of the neurons of the ideal flowers) not only as points in the coordinate system but as **vectors**. You will ackquire needed vectors joining the point with the beginning of the coordinate system. That is exactly the way program **Example 01b** will work. It – similary to the previous one – can be found in the **Start**[2] menu. Playing with this program notice the preceding facts:

- The value of the output depends mostly on **the angle** between the input vector (representing the input signals) and the weight vector (the ideal object accepted by the neuron). It is illustrated in the figure 4.9.

- if the angle between the input vector and the weight vector is **small** (both vectors are located next to each other) – the value of neuron's output is **positive** and high

- if the angle between the input vector and the weight vector is **big** (both vectors create angle greater than 900) – the value of neuron's output is **negative** and high

- if the angle between the input vector and the weight vector is **close to 900** – the value of neuron's output is low and **neutral** (near zero)

- if the length of the input vector is far more smaller than the length of the weight vector - the value of the neuron's output is **neutral** (near zero) independently on the direction of the output vector**;**

All of the described characteristic features of the neuron's behavior you can test on your own using the **Example 01b** program. Although the pictures produced by it, aren't so graphically worked out, as the figure 4.9, they should be easy to understand. They would be the convenient basis to collect your own experiments letting you to learn and remember what a neuron actually does.

The work with the program is quite easy. The only thing you need to do is to click in the area of the located on the left chart. First, click it with the **right** button to set the location of chosen point corresponding to the neuron's weight factors (see Fig. 4.10). You will be shown the point and its coordinates. Of course you can change it at any time, clicking again in the other part of the chart or modifying the coordinates manually – the same way you did it in the **Example 01a** program. Now click on the chart with the **left** button, to locate the position of 'the flower' and watch the answer of the program. If the neuron likes the flower (on the right there is the value of the output signal so you can easily get to know what your neuron thinks of the flower) then the appropriate point is marked on the chart with **red** (like a mountains on a map – see Fig. 4.11). If the quotient is negative – point is marked with **blue** (like a seabed on a map – see Fig. 4.12). When the reaction of the neuron is neutral – corresponding point is marked with **light blue** (Fig4.13). After some time you will be able to imagine how the areas corresponding to the decisions in the input space look like. To picture them easier you can try to drag the mouse pointer over the chart with one of the buttons pressed to observe the results changing.

2 If you haven't installed the example programs yet – do it – the fun is about to start right now

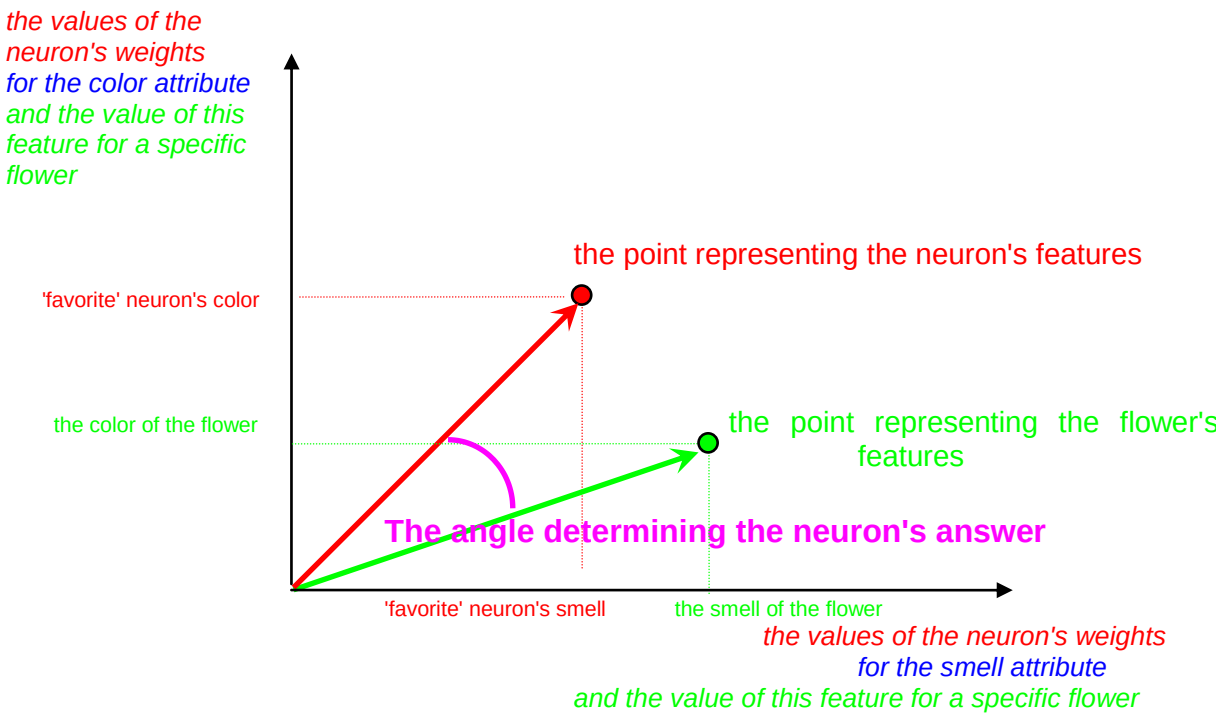


Fig. 4.9. Mutual position of the weights vector and the input signals vector – as the factor determining the value of neuron's response

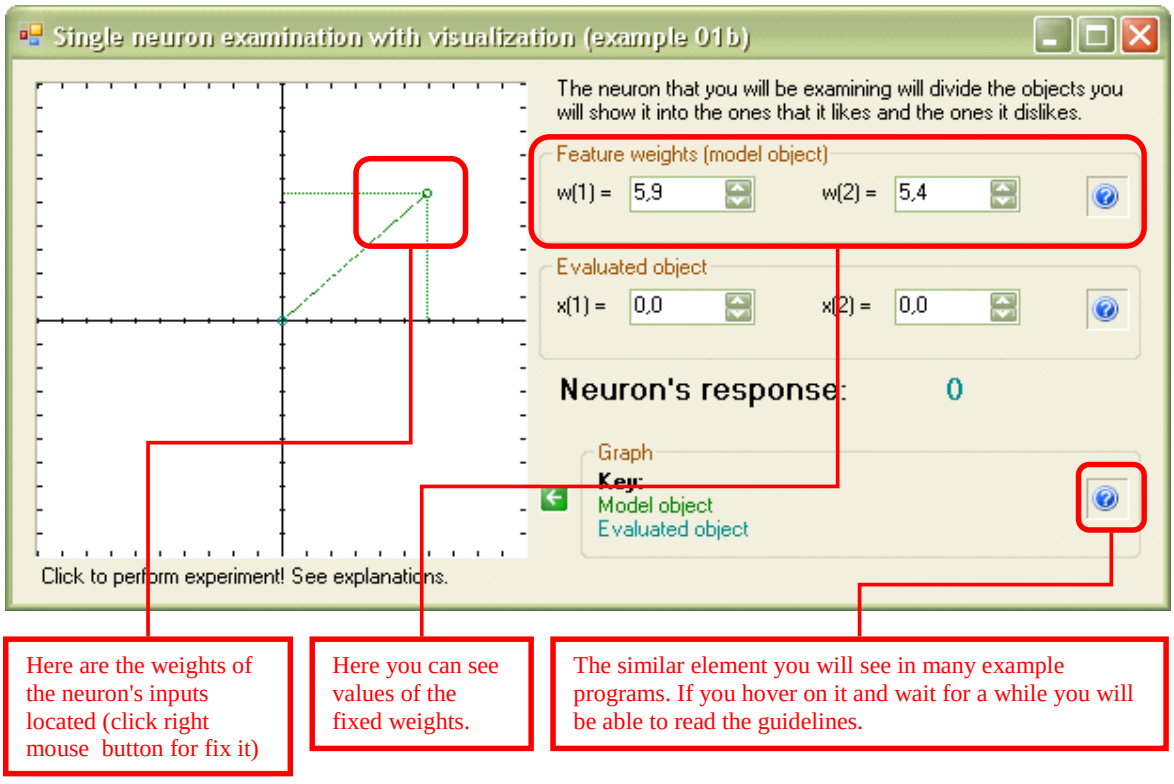


Fig. 4.10. The window of the **Example 01b** program with the model object marked

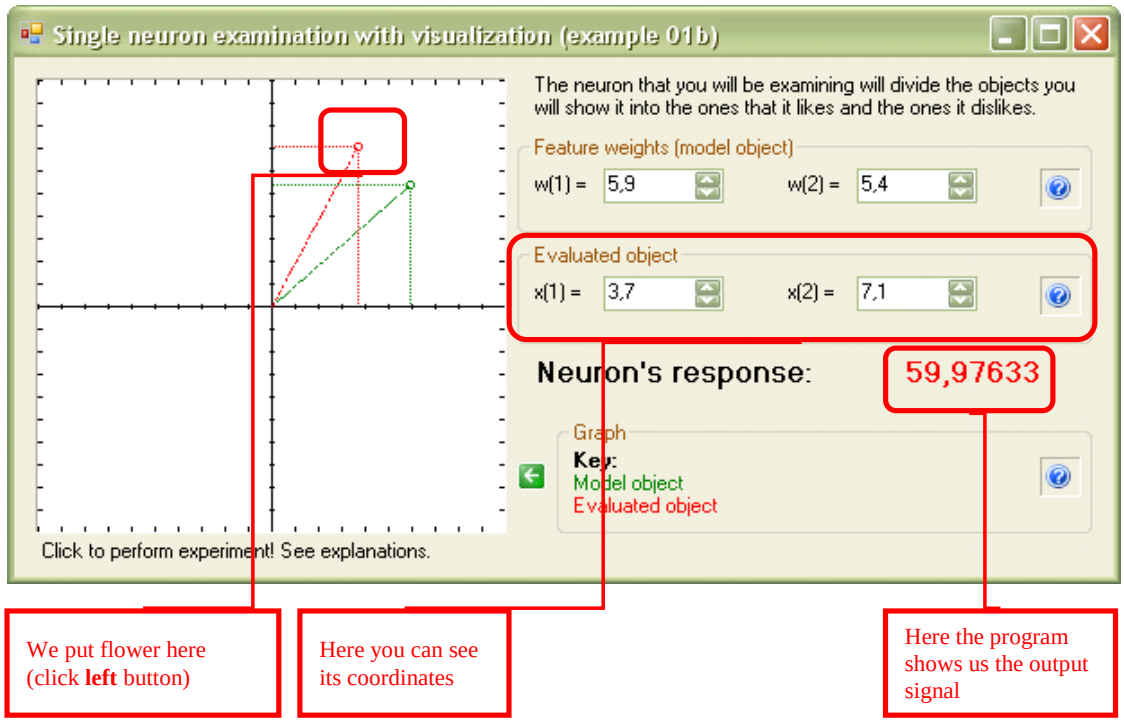


Fig. 4.11. The presentation of the input vector location for which the output signal is positive

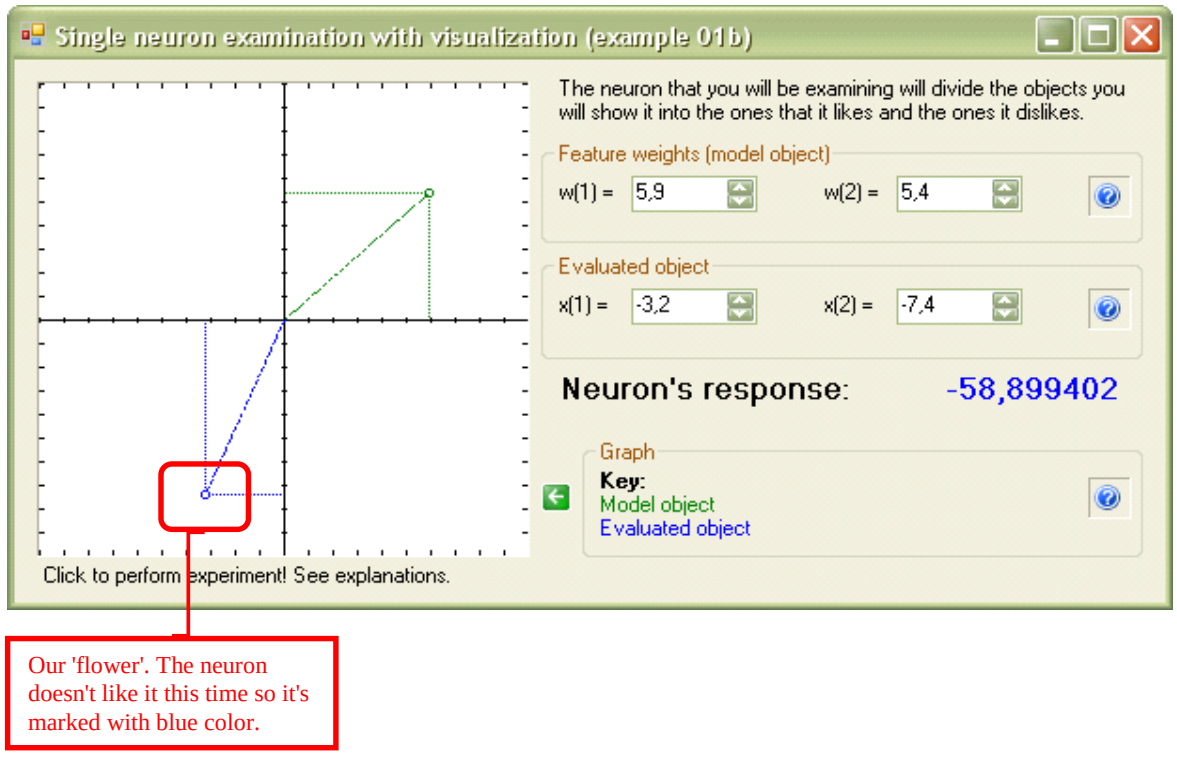


Fig. 4.12. The presentation of the input vector location for which the output signal is negative

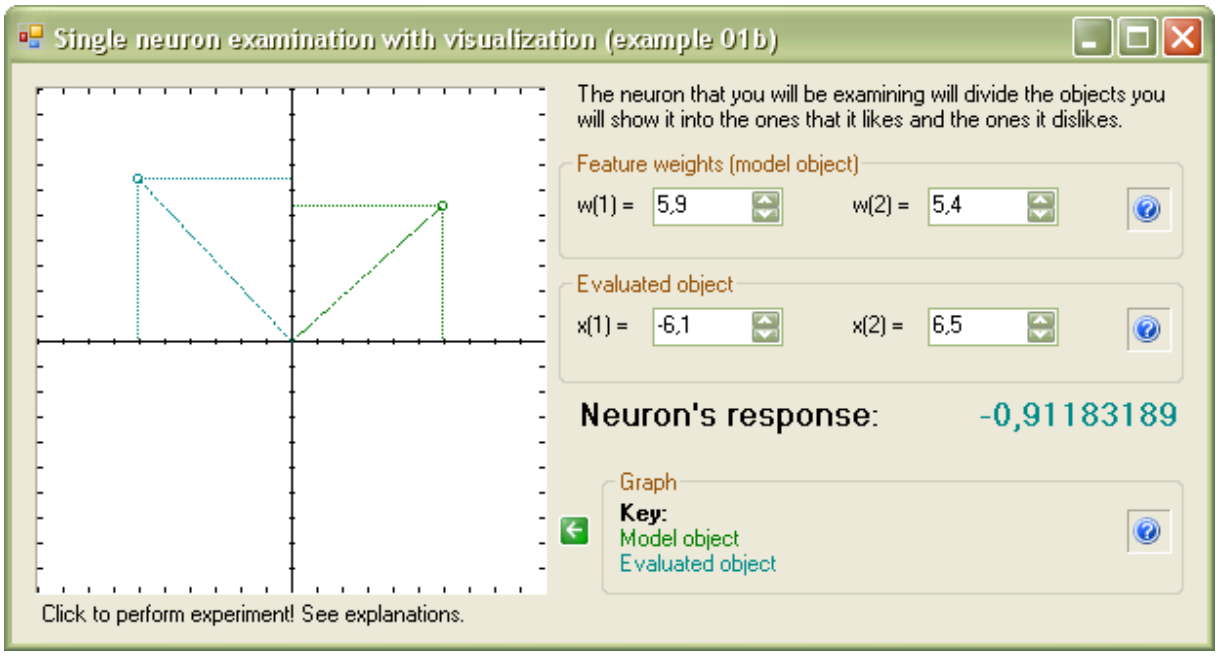


Fig. 4.13. The presentation of the input vector location for which the output signal is neutral

4.4. How to manage with the bigger amount of the inputs of the neuron?

(Translated by Krzysztof Kajdański; krzysiek.kajdanski@wp.pl)

The examples above are nice and simple because they concern only one neuron with two inputs. In real application of the neural networks you typically have to do with task where many inputs are considered (because the solution of the problem you try to solve with the neural network often depends on the huge amount of the input data) and there is a need to apply many cooperating neurons to gain the level of compliance of the network appropriate for the current problem.

That's the point in the neural networks where jokes ends and problems begin. You cannot draw this (because nobody is able to imagine points in the input space having more than ten dimensions!) and that's why everyone thinks that it must be very hard. However – it doesn't have to be that way. I'll convince you in a minute. Just try out the program **Example 01c** (see Fig. 4.14)

The program asks to enter the number of the neuron's inputs. You can accept the default value (5) and click Next. Then, as previously in the **Example 01**, you should input the weights, which define the model of the signal, on which your neuron should react positively. Fill in the column marked as **w(i)** (Fig. 4.15). Then you enter the values of the input signals in the **x(i)** column and then program will calculate the output. Really simple, isn't it?

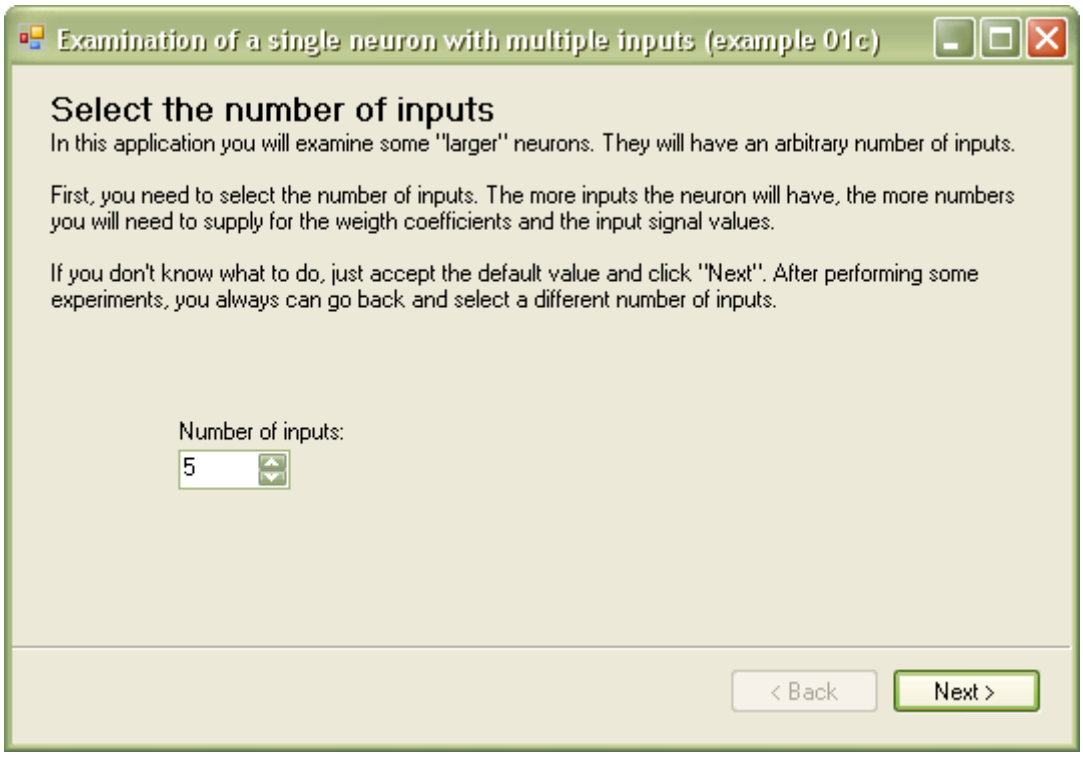


Fig. 4.14. The beginning of the conversation with the **Example 01c** program

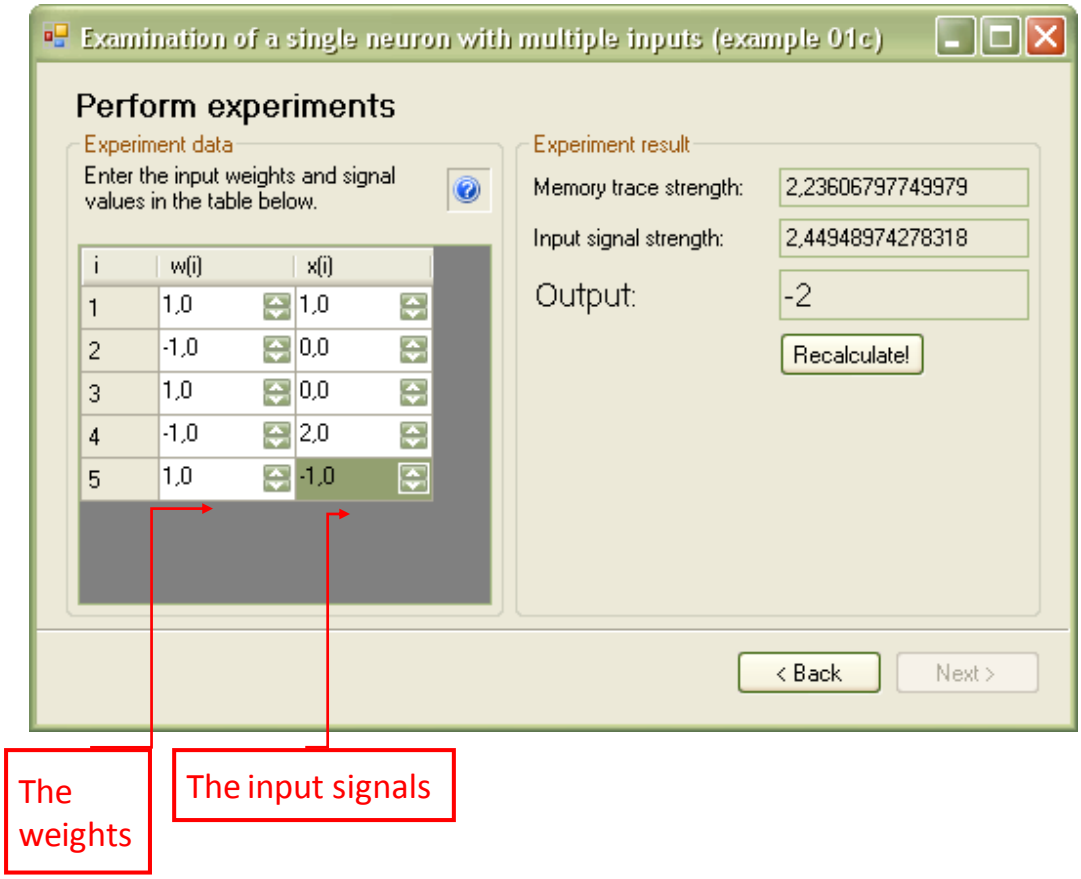


Fig. 4.15. The sequent part of the conversation with the **Example 01c** program

Don't worry that the program tells you something about some strength. You will get to know, what you can use it for in a minute.

During experiments with this program you can easily notice, that even when we work in the five dimensional space of weights and signals (that's a kind of space even Albert Einstein didn't work with – he stopped on the four dimensional continuum) – we can easily foresee what neuron will do. When while setting the coordinates of the input signals vector, you set them to be similar to the corresponding weight vector coordinates – these values are obligingly recollected by the program – you will get high value of the output signal. It is quite obvious: similar values of weight vector and input vector means that these vector lay close to each other and this – as you know from previous consideration – is a condition of getting the high output signal. On the other hand, if the values of the coordinates of input signals vector are set similarly to the values of the weight vector – but with **the opposite sign** - you will make the input signal vector pointing another direction than the weight vector. Of course the effect of this action would be the neuron generating strong **negative** output signal. Finally, if you don't try to keep any accordance between the input signals and the weights – there is a chance that your multi-dimensional neuron will remain haughtily indifferent towards the input signals producing output signal with very small absolute value.

During experiments you will notice that high values of the output signals are returned in two situations: when input signals **correspond to the weights of the neuron** (that's what we expected) or just by entering **huge** input values, where weights are positive. The first way of obtaining the high output is intelligent, sophisticated and elegant – well, whatever, since the same (sometimes even better) effects we can get using the brute force and the method number two. Therefore during entering the input signals you should try to make them of the same strength (using the parameter given by the computer estimating the actual strength). Only then you will be able to correctly interpret and compare your results. Similarly comparing the behavior of two different neurons (with different weight values) for identical input signals – they should tend to have the same value of memory trace strength – it is just the length of the weight vector. In the network with the bigger amount of neurons the meaning of the strength of the input signals radically decrease, when – stronger or weaker input signals – get to every neuron and **the difference** of the signals from different neurons decide about the effect: these better 'tuned' to the input signal and those worse 'tuned'. We will focus on that in a minute. However when we consider only one neuron – dissimilar values of input signals can make results harder to interpret – that's why we should agree on choosing the input values for examined neuron in a way, when the sum of their squares is (estimating – high precision is not needed here) between **5 and 35**, for instance. Because program calculates the strength as a square root from the sum of squares of the coordinates (that's the formula for the length of a vector), let us agree that the strength of the signals should be between square root from 5 and square root from 35, which is somewhere between 2 and 6.

Why should we choose these values? Because during my work with the program I found out, that when entering small random integer values for five inputs of the neuron – you will get (more or less accurately) values from this range. However if you prefer, you can choose any other value and it is going to work, but remember to keep with it. The same limitation should be applied to the values of weights (it will turn out to be useful, you'll see!). Thanks to it, it will be easier to check, if the input signals 'match' the values.

4.5. What does the simple linear neural network act like?

(Translated by Krzysztof Kajdański; krzysiek.kajdanski@wp.pl)

Let us assume that there is more than one neuron and they are organized in a network, which is single-layered (it means that neurons are not connected with each other, only with input and output – see Fig. 4.16).

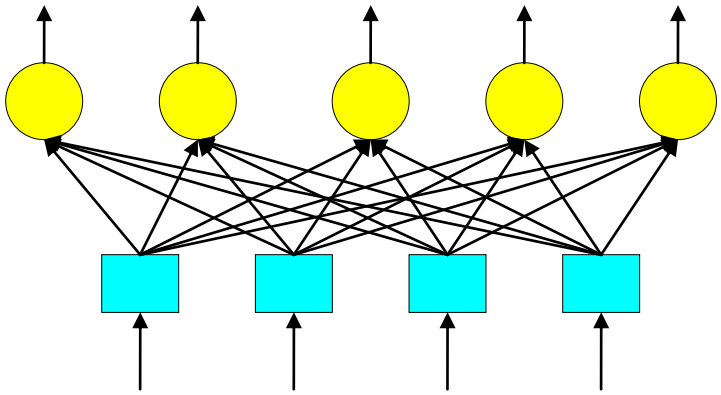


Fig. 4.16. The structure of the single-layered neural network

Input signals you will apply to the network will be entering every neuron, output signals of the neurons will be treated as an answer of the whole network for the given task.

How does it work?

Every neuron has its own set of weights, which make it ready for recognizing the characteristics of the input signals. Every neuron has different weights and recognizes different patterns of the input signals. It means that when you enter the input signals every neuron will calculate its output signal **independently**, which will turn out to be high for one neuron and small for others (because this one particular neuron has recognized its pattern). Analyzing the output signals you can easily get to know, what pattern network 'suggests' (basing on observation of the highest output value of neurons), and you can see how much the network is 'sure' of its decision – comparing the output signal of the 'winning' neuron with the signals generated by the rest of the network elements.

Sometimes this feature of a network, allowing to detect uncertain situations, is the most useful in practice, because there is nothing worse than algorithm that makes very certain decisions basing on incomplete data and estimated methods of concluding. If such an algorithm is used wisely – it may be useful of course. However – people use to trust computer more, the less they understand them. So if we imagine a fool equipped in too arbitrary expert system – the vision of a monkey with a razor is in comparison to that horror a child's play.

4.6. How to construct a simple linear neural network?

(Translated by Krzysztof Kajdański; krzysiek.kajdanski@wp.pl)

I'll show you another simple program. This program named **Example 02** will help you start a simple play with a very small neural network. I encourage you though to write by yourself a more complicated program solving one of your real problems.

Network described in the **Example 02** program recognizes animals. There are three categories of the animals (**a mammal, fish, a bird**) and therefore network contains only three neurons. Recognitions are made basing on five features, that make every neuron to have five inputs. On these inputs we send the following information:

- how many legs does animal have,
- does animal live in water, can it fly,
- is it covered with feather,
- does it hatch from an egg.

For every neuron I set the values of weights in such a way to match the pattern of an appropriate animal. Neuron number one, which is supposed to recognize **a mammal** has the following weight values set:

- 4 = mammal has 4 legs,
- 0.01 = mammal sometimes lives in water (seal), but it is not typical for it
- 0.01 = mammal sometimes flies (bat), but it is not typical for it
- -1 = mammal has no feathers
- -1.5 = mammal is viviparous and it is a major feature of it

The weights of the neuron number two recognizing **a bird** are set in the following way:

- 2 = birds has two legs,
- -1 = a bird doesn't live in water (a duck only swims on a surface!),
- 2 = a bird usually – which is important to it – can fly (exception: ostrich),
- 2.5 = a bird has feathers and it is a major feature of it,
- 2 = a bird hatches from eggs.

The weights of the third neuron identifying **fish** are set in the following way:

- -1 = fish has no legs

- 3.5 = fish lives in water and it is a major feature of it
- 0.01 = fish generally cannot fly (there are flying fish!),
- -2 = fish is never covered with feathers or nothing that resembles them
- 1.5 = fish generally hatches from egg, which is not as important as in bird case, because there are exceptions (viviparous fish);

Program after start prints out on a screen information about previously described weights for every input of every neuron (Fig. 4.17) and allows to perform quite amusing experiments, which I describe in the following sub-chapter.

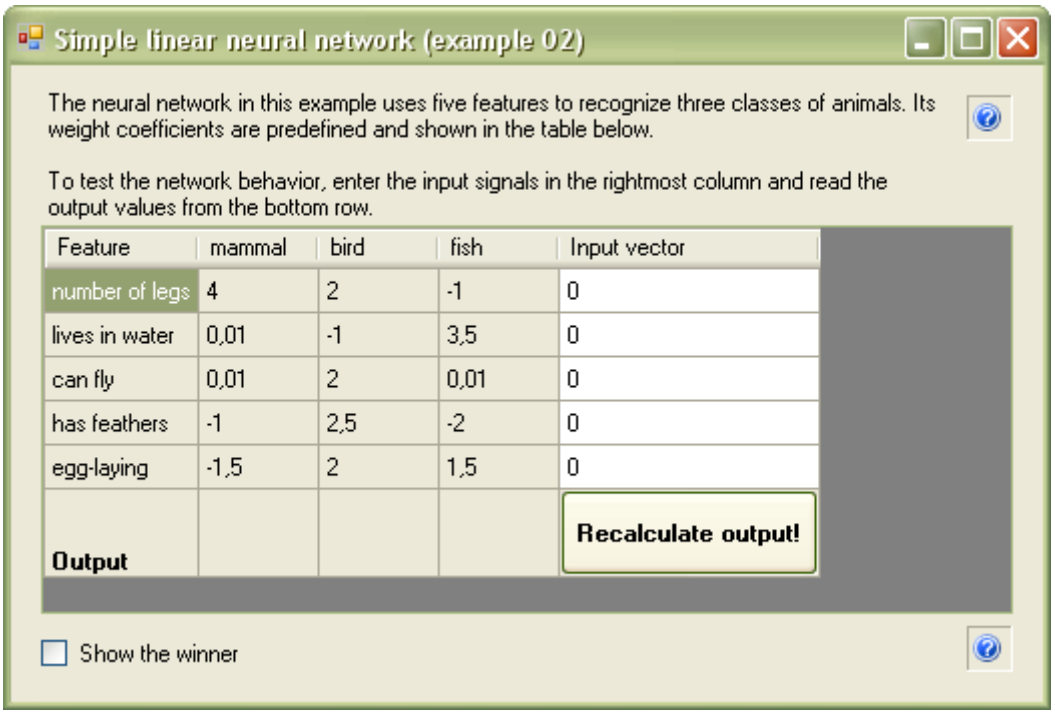


Fig. 4.17. The beginning of the work of the **Example02** program

4.7. How to use the described neural network?

(Translated by Krzysztof Kajdański; krzysiek.kajdanski@wp.pl)

As I have mentioned before, the program presented previously, assumes that network has three outputs associated with the recognition of the three kinds of objects (animals): mammals, birds and fish. Because a network is single-layered – it means that it contains only these three neurons, although you will learn in the future about networks in which the number of the neurons is much more higher than the number of outputs. Thanks to the simple construction of the network it is very easy to reconstruct it to any number of the outputs.

You enter the input – in our example – only five signals, corresponding to some features of recognizing objects. Obviously you can increase this number in a trivial way, if in your problem that you would like to solve with the network, there is a greater amount of the input data. All input

signals are connected to every neuron, according to 'the lazy rule' – if you don't feel like thinking which input signal influences which output signal – it is the best choice to connect everything to everything else. In practice of neural networks you usually do so, so it is the best for you to get use to this convention.

It would be good, if you thought a little bit, **before entering the input signals**. Amongst them there are ones containing numeric information (info about how many legs animal have) or Boolean information (info concerning if an animal lives in water or flies, if is covered with feathers, or is viviparous). You need to consider in what way you will represent logical values in a network, because neurons operate – as I have mentioned previously – on **values** of the signals, not on a symbols (just like **true or false**). Because it is not easy – let me share a reflection.

If you are an computer scientist (as I suspect you may be, as you are interested in neural networks) you may suspect that the idea of **true and false** can be expressed in binary form: **1** as for true, **0** as for false. And if you are the a Great Computer Scientist, especially from that kind that use Assembler, have dreams about microprocessor registry, hexadecimal memory records and Java applets – then that kind of relation is obvious, total and the **only** right.

Hmmm...

I have something to confess. Entering in the reign of the neural networks will make you modify your habits. Here is where you have to do with it for the first time, but not for the last.

Remember then that zero in neural network is quite a stupid signal to transfer (one from this kind which is absent, that means it brings no new information). It is due to way neurons work, multiplying signals by weights and then summing it – multiplying by zero results always in the same way – independently on the value of neuron, which represents the inner knowledge of a neuron!

Deciding on using zero as an input signal you deprive yourself of some possibilities of influencing on the network behavior, which is not smart or necessary. That is why in my program I use convention, to using which I encourage you: true marked with **+1** signal and false with **-1**. Such bipolar signals are the best in fulfilling their tasks.

Additional advantage of a bipolar neural network is a possibility of using **any** values of input signals during inputting the data, and that may reflect the convictions of a person using a network about an importance of some messages. When inserting data for **a cod** you can consider as crucial the fact that it lives in water and input +2 instead of +1 (you can interpret that as a yell 'of course, it does!' instead of calm 'yeah, it's true'), and for other cases you can input values smaller than one (for instance – for flying fish and the question 'can it fly?' you may doubt if you should answer +1, as for an eagle – in that case you can put uncertain and full of doubts +0.2, which may be interpreted as 'sort of...'). There's a lot more of possibilities, for example signal corresponding to question 'has tail' for snake should be of value '+10' ('*only!*' :-)).

Since you know how to input network input signals – try to perform a few simple experiments with me. Input data for a few randomly chosen animals and check if the network recognizes them

correctly. You can notice that in network consisting of many elements, the normalization of the input signals (caring if they have the same signal 'strength') is not as crucial. You will get to know that, by giving input values – for instance for **a fox**:

- 4 (legs),
- -1 (doesn't live in water),
- -1 (doesn't fly),
- -0,9 (not covered with feathers - unless it just caught a goose...),
- -1 (viviparous);

as the values

- 8 (has 4 legs, but it is so important that it is worth to be counted twice),
- -6 (hates water, especially in barrels hidden in the ground!),
- -3 (absolutely doesn't fly, even if it wished to sometimes...),
- -5 (ginger fur is an important feature),
- -9 (doesn't lay eggs and doesn't tolerate similar suspicions!).

During the testing of the program it is good to notice that it can not only recognize correctly typical situations (it will classify properly every mammal, fish and bird – see pictures 4.18 4.19. 4.20), but it acts quite well in strange situations – for instance it will recognize as a mammal a seal, a bat or even a platypus (it is a weird mammal from Australia – it hatches from eggs), and as a bird – a non flying ostrich for instance. (Even a flying fish is recognized with no failure!) Try it out!

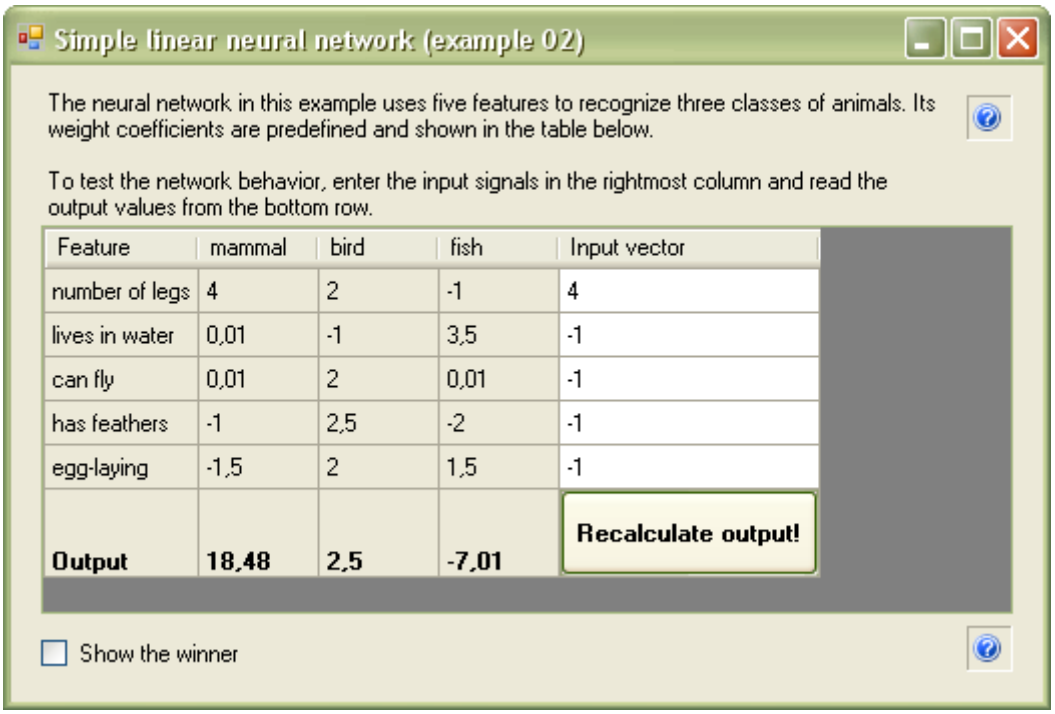


Fig. 4.18 Recognizing a typical mammal using the network.

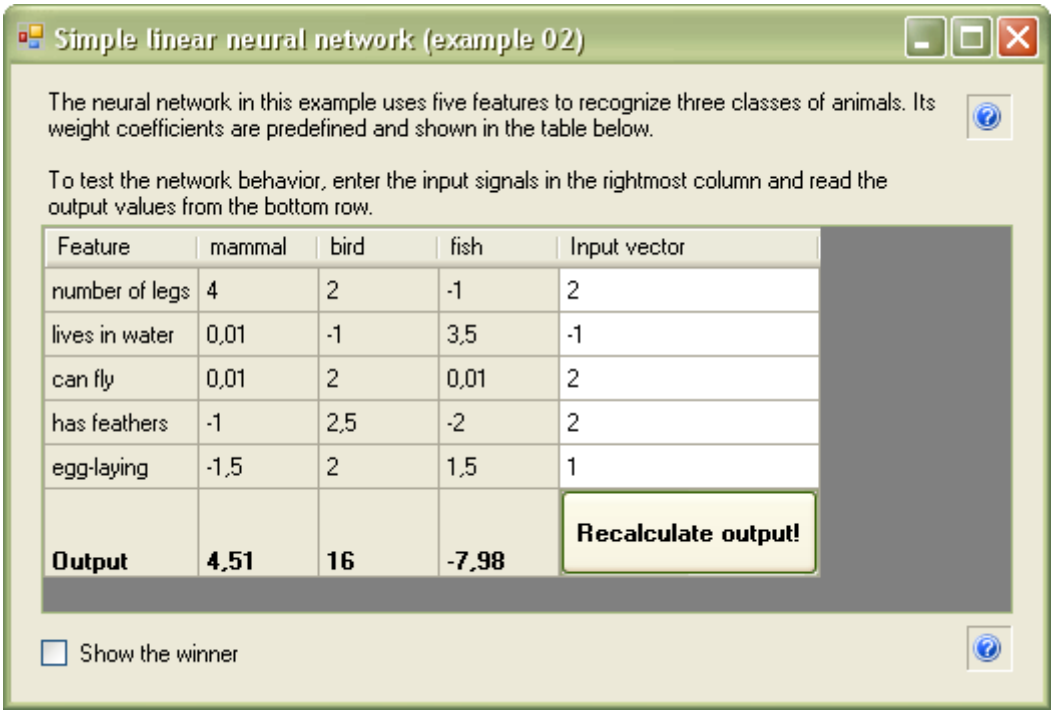


Fig. 4.19. The work of the program during the bird identification.

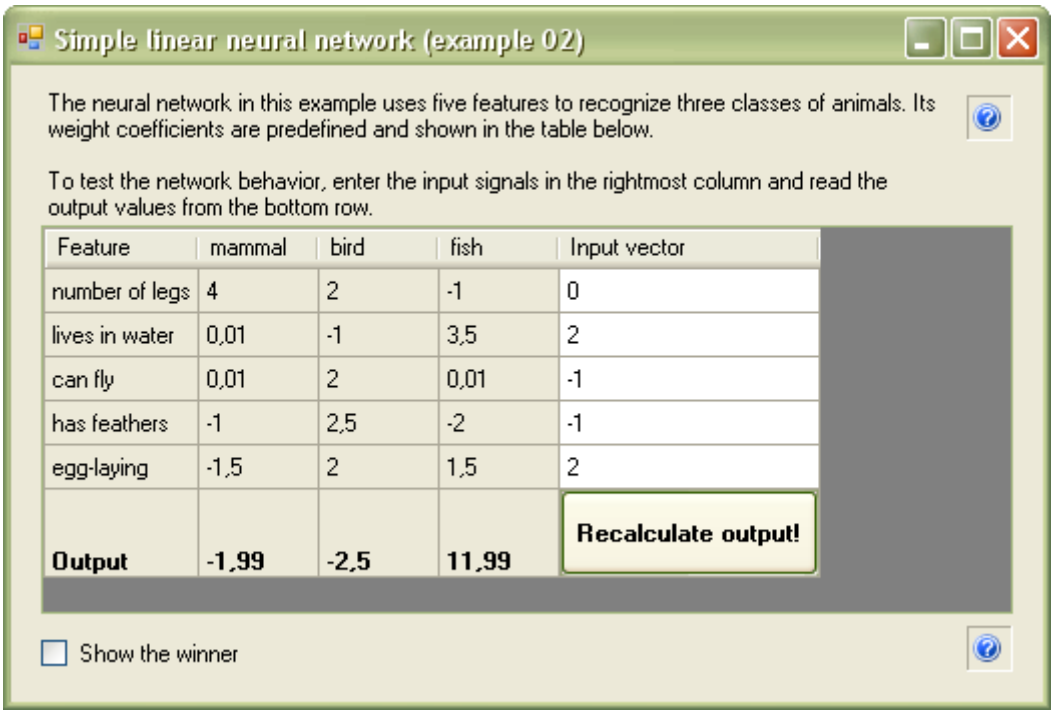


Fig. 4.20. The work of the program during the fish recognition.

However it will be puzzled if we show it a snake (it has got no legs, lives on a ground and hatches from eggs???). Then every neuron disgusted announces that something like that isn't a decent animal (every output is negative), which, in context of our rules of classification, makes sense.

Described network is very primitive and makes sometimes mistakes, for instance repeatedly recognizes turtle as a mammal (it has 4 legs, lives on a land, but hatches from eggs, so there should be some doubts....), or even considers lungfish as mammals (they live during drought on a land – please check it, if you don't believe me) – but that is how it is with the neural networks – they can be wrong in such a nice way – if any of you haven't made ever a mistake – let him throw the stone at the monitor first!

4.8. Why and what for there is rivalry in the neural networks?

(Translated by Krzysztof Kajdański; krzysiek.kajdanski@wp.pl)

In practical application you sometimes use additional mechanism of 'rivalry' between neurons, which in some applications allows a network to perform better. It is possible to observe how the competition network works when we apply to the previously discussed network-recognizing animals – an element comparing every output signal and selecting a winner. A winner is a neuron with a highest value of the output signal. Selecting a winner may have consequences (for instance you can allow only one neuron to learn – as it works in the Kohonen network, which I will describe later) but in most cases it is used to polarize the output signals in the network – for instance only the neuron that is a winner can output its signal, every other output is zeroed. This rule is called **WTA** (winner takes all) and allows to analyze the way the network works easier (especially when it has many outputs), but is sometimes dangerous as well (as mentioned previously).

The best thing to do is to try it out, introducing the element of rivalry to our program simulating the recognition of animals. For that purpose run program **Example 02**; this time check the box signed as **Show the winner**. As a result after processing the input data the program will mark with the red color the neuron, whose the output value is the highest and announce an unambiguous verdict. The example of that action is shown in the Fig. 4.21.

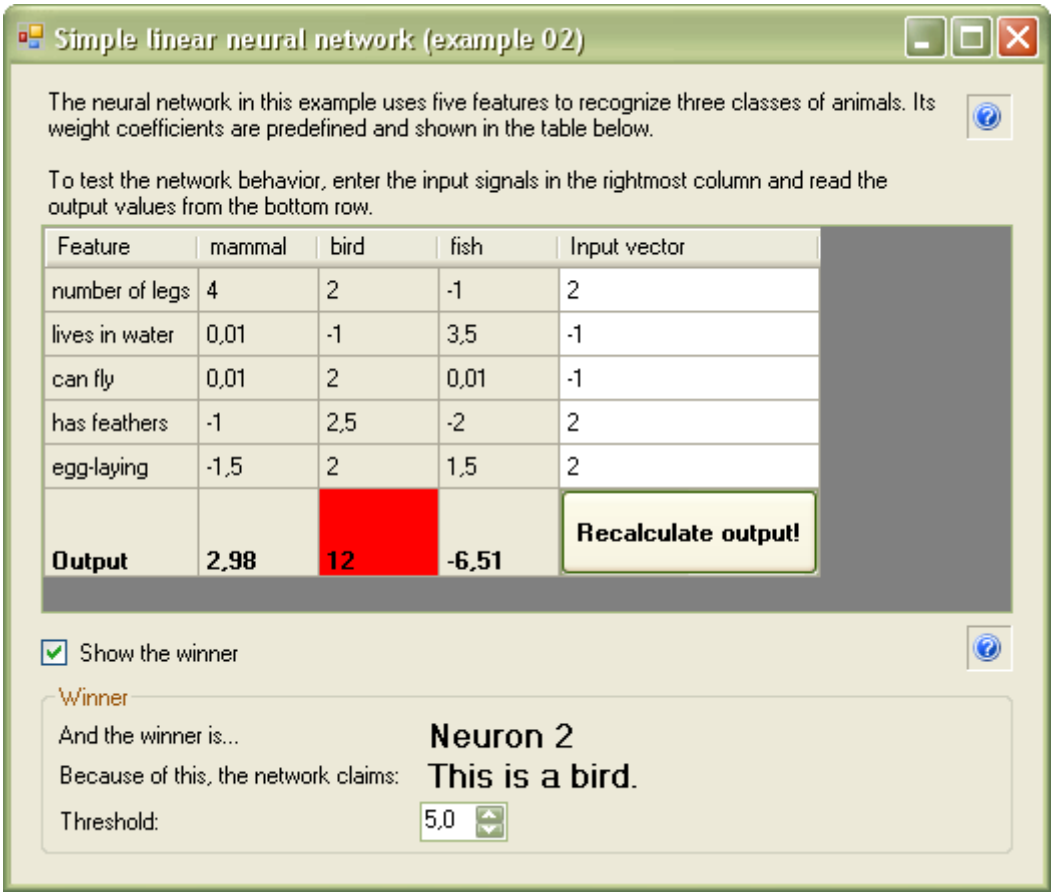


Fig. 4.21. The example of the program **Example 02** operating with the rivalry option enabled

Notice, that during neuron competition we assume that only the positive value is a basis to make a decision. If every output signal is smaller than value marked in program as threshold (you can set it as you wish) – the output signal should be a **no recognition** signal.

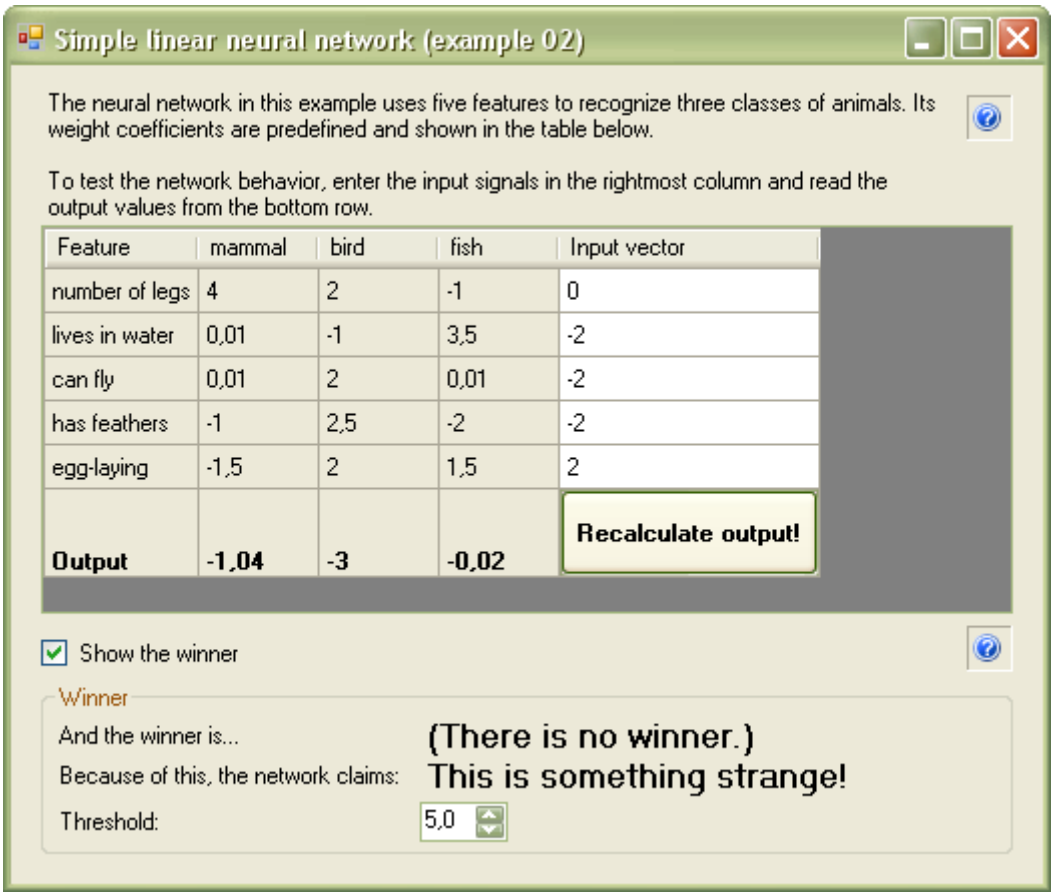


Fig. 4.22. The network with the rivalry option enabled trying to recognize a snake

I encourage you to use program **Example 02** to try to perform experiments on the described competition network. You will find out that this network – contrary to previously defined 'regular' linear network – has few nice properties: answers more categorically and, what's more, in a form of a text, not as numbers which require additional interpretation. The competition network has a limits as well, which you learn about during next chapters of this book.

4.9. What are the further possibilities of an application of the neural network?

(Translated by Krzysztof Kajdański; krzysiek.kajdanski@wp.pl)

The purpose of the network described previously was to recognize some sets of information treated as a set of features of recognized objects. However it is not the only application of the simple single-layered networks consisting of the neurons of the linear characteristics. The networks of this type are often used for other purposes, such as the signal filtering (especially as adaptive filters with properties changing in dependency on the current needs). They have also numerous applications in the signal transformation (speech, music, video, medical signals - EKG, EEG, EMG) – for example they can extract a spectrum of a signal or arrange the input data using the *principal components analysis method.* I have mentioned only few example tasks, but the application of every network – even as simple as the one described previously – can very differ.

All decisions of the network are made by set of weights for every used neuron. By setting the weights differently, we change the way the network works. In the same way by changing a program we make computer work differently. In examples described previously we used the set of weights chosen arbitrarily (you had to figure out on your own what weight value should every neuron have), which can be interpreted as a change of the work program for the network. For a few neurons contained in the network, as well as for simple and quite obvious interpretation of the weight factors contained in it (as it was with the example of the animal recognition) – that kind of a 'manual programming' of a network can be used and have good results. However in practical applications networks have many elements and then the role and tasks of a single neuron are hard to follow, and that is why more useful networks choose the weights on their own in the process of learning. Therefore in next chapter we will devote our attention to the most important thing – the aspect of learning the networks.

4.10. Control questions and issues to solve.

(Translated by Krzysztof Kajdański; krzysiek.kajdanski@wp.pl)

1. What properties do the neuron's weights and the input signals need to have to make the output signal:
    - strong and positive?
    - strong and negative?
    - close to zero?
2. How can you make the neuron to favor one of its inputs (for instance to make the color of a flower more important to him than its fragrance)?
3. How can you interpret the positive and the negative values assigned to every input of a neuron?
4. How can you interpret the positive and the negative output signals of a neuron?
5. Does neuron having **all** the input values negative have to have the negative output signal?
6. Is there any limit of an amount of the neuron's input signals?
7. Check, if the network modeled in the program **Example 02** recognizes **a dolphin** as a mammal or as fish?
8. What can be achieved thanks to the rivalry in the neural network?
9. Check what animal group **a bat** will be assigned to by the network modeled in the program **Example 02.**
10. Check, if the network modeled in the program **Example 02** recognizes that **dinosaurs weren't** mammals, nor birds, nor fish (they were reptiles and that's the category we

don't have). Which known to the network animals will dinosaurs turn out to be the most similar to?

11. Is there always in the competition network a winner? Is that good or bad?

12. **Task for advanced:** In the neural network recognizing the animals add additional classes of animals (for example predators or herbivorous animals) and extra data describing the animal's features (the sharpness of teeth/bill or ability to fast movement).

5. Teaching simple linear one layer neural networks

5.1. How built teaching data?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

In this chapter you have to find out about how to teach the simplest network .

I know that this matter has been discussed, but theoretical knowledge (which is result of reading algorithm description) is something completely different than practical knowledge (based on doing something on your own and seeing the results).

While describing the experiments shown here we will still talk about linear networks, because in the previous chapter you have made researches on linear neurons included in networks. You have also built your first simple linear network and you have conducted some research on it – that is why, you should know how such neurons and such networks behaved.

In the next chapter (number 6) I will also show you **nonlinear** networks and then you will get to know in practice, how this network is built and what kind of properties it has.

Therefore allow me to describe in the next chapter (on the basis of differentiation) what is meant by the adjective **linear** (so often emphasized and repeated with names of networks in this chapter)

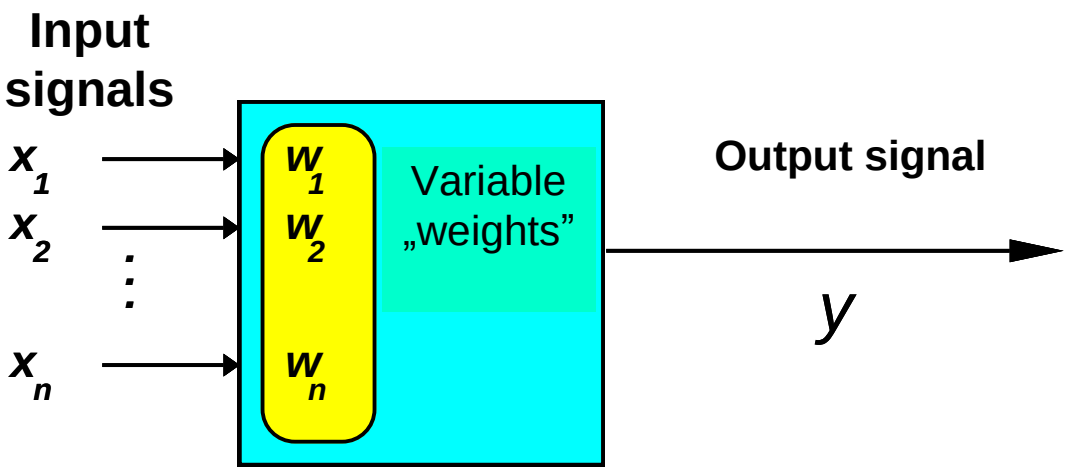


Fig. 5.1. Structure and basic elements of artificial neuron being taught.

Now take my word for it that linear networks are much simpler than nonlinear, and will be easier to give them tasks while teaching and to check the results.

Moreover we will start the game from the easiest of the easiest tasks – from teaching a single neuron.

To achieve this, use program **Example 03** which will let you conduct simulation research and automatic teaching of such simple linear neuron (fig. 5.1).

However, before you put data in your computer, before you try out and (of course!) improve my teaching program, you need some introductory remarks.

During the experiments based on research on the work of a single neuron and the entire network (in the previous chapter) you have used the manual possibility of putting all the needed signals, which gave you full control over the experiment.

Then this method was easy, nice and comfortable.

The moment you have tried to teach a bit larger network and more difficult tasks, it won't be a piece of cake any more.

Sometimes hundreds or thousands of experiments will be needed before something reasonable begins to appear out of the initial chaos.

Of course, we can imagine the masochist who writes in the same data several thousands of times in order to teach the network the giving of the correct task solution. I assume, however, that among readers of this book such madmen are definitely in minority.

Therefore from the beginning let's assume that teaching process must be based on a set of teaching data which was created and saved on a computer. The set should include input signals (to all the neurons in the network) and the models of the correct (required) output signals, which the teaching algorithm will use to confront the real network behavior.

In my programs the format of records of the set of teaching data will be as follows:

comment (helps to observe what is going on)

set of input signals

set of models of the correct output signals

I will assume here, that a set of input signals will include five-element-vectors (portions including five signals for five inputs of the analyzed neuron), while there will be only one model of the correct output signal (because you can temporarily use only one neuron).

These network parameters are placed in the first line of the teaching sequence file.

A teaching sequence can be as long as you wish (to be honest, the more examples with correct solutions you show to the network, the better). Hence, it happens that the file including the above mentioned sets of information can really be very big.

To start with, for a program which teaches one neuron, I recommend the use of the following file:

5, 1

A typical object that should be accepted

3, 4, 3, 4, 5

1

A typical object that should be rejected

1, -2, 1, -2, -4

-1

An untypical object that should be accepted

4, 2, 5, 3, 2

0.8

An untypical object that should be rejected

0, -1, 0, -3, -3

-0.8

The above presented text is a teaching file for **Example 03** program which is to be found in a file called **Default teaching set 03.txt**.

Example 03 program has information about that, so it will offer you to use that teaching file at first.

Of course you can create your own, completely different file with data for teaching a network and connect it to the program described below.

In this way you can teach your neuron to recognize completely different sets of target standards or you can force it to built a **model** of some occurrence (I mean, force it to approximate some relationship between input signals and output signal – for example, as a result of medical observations or physical measurements).

But remember that this neuron is **linear,** therefore it is only able to learn how to transform signals similarly to, for example, correlation methods or multidimensional linear regression (these are names of mathematical methods used by scientists for statistical description of the experimental research results), but is not able to learn more!

You will get to know the wiser neurons and more universal **nonlinear** networks, but not right now.

However I would suggest to use my program at the beginning.

Of course, as you will try this program out and see that everything works correctly – you will be able to enjoy the work with your own data without limitations, but with the knowledge, on what kind of data you teach your network – I will describe beneath, what you see during this teaching and what is result of the fact that you see just that, and not something else.

5.2. How can we teach one neuron?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

At the beginning lets load **Example 03** program and try to start it.

You can see, what the program is doing: it is loading the data from the file (which is called, as we estimated, **Default teaching set 03.txt**) and first trying to classify objects correctly by itself. Data of the objects are recorded in file in the right order (in the form of input signals for neuron).

If it does not work – and a neuron can notice this fact itself, because the **Default teaching set 03.txt** file has the models of the correct answers recorded by the teacher – then the program modifies (according to the detailed description in one of the previous chapters) the weights of modeled neuron.

In this way it teaches that neuron to perform better when given a defined task.

During simulated teaching, on the screen you can observe the progress of teaching process step by step, and watch how the weights change and see the errors.

At the beginning errors are large – of course. (fig. 5.2)

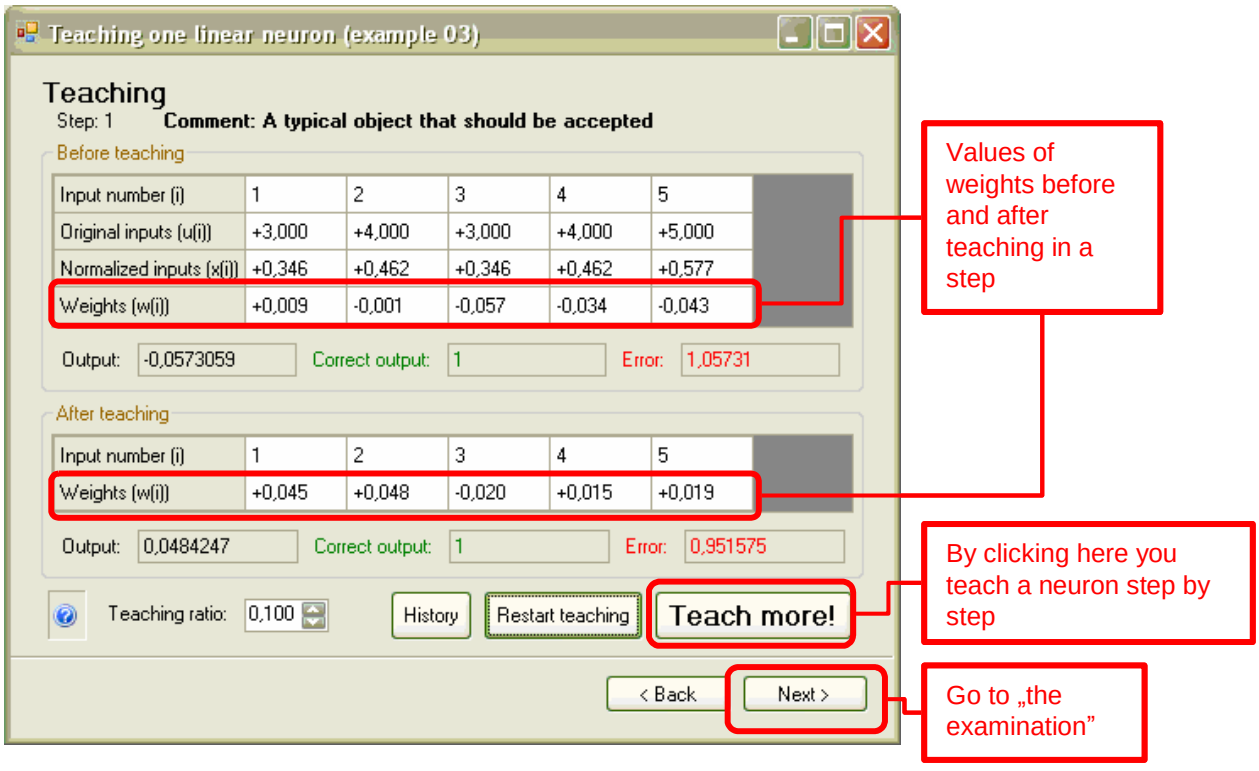


Fig. 5.2. Beginning of the process of teaching a neuron

If the network is being “tested” by you at the moment (you can go to this phase every time during teaching process, by clicking **Next** button), you may check what knowledge it has.

However at the beginning of the teaching process you will see, unfortunately, that your hope, that a neuron should already know the shown objects, will turn out to be deceptive (see fig. 5.3).

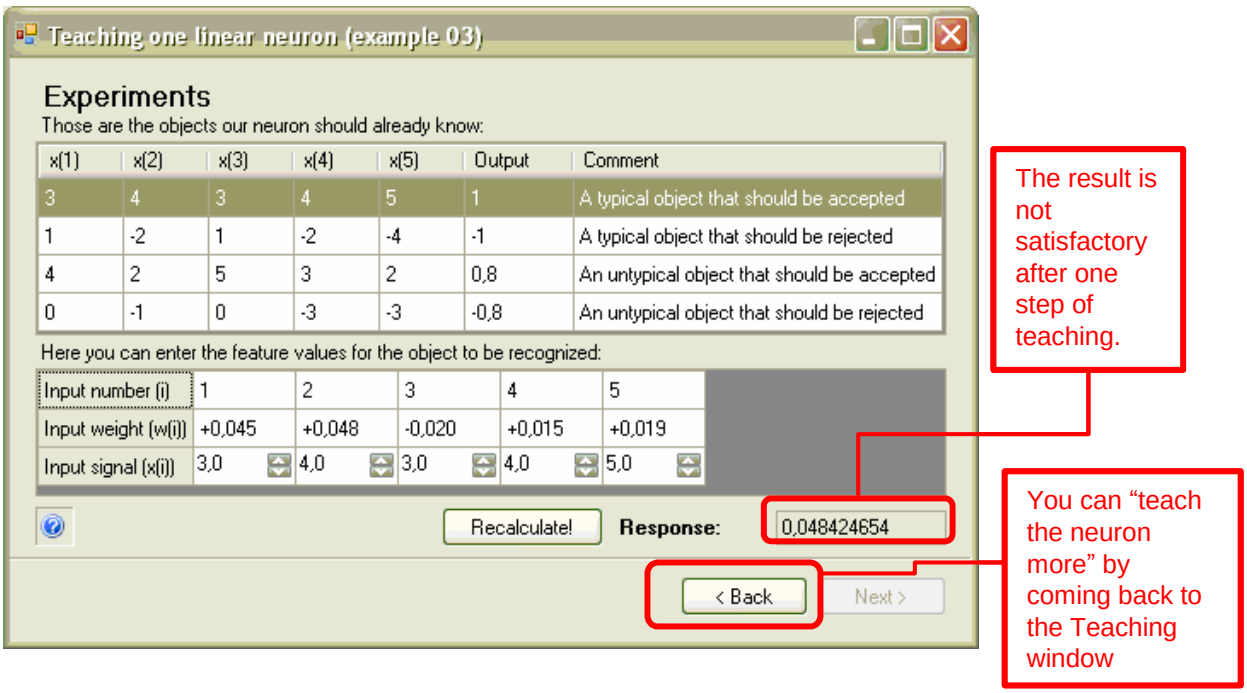


Fig. 5.3. Unsuccessful result of examination in case of poorly trained neuron (shown object was weakly recognized by the network)

In order to get the neuron working in a reasonable way, it is necessary to teach it a little bit more.

By clicking **Back** button let’s come back to previous **Teaching** window and teach neuron patiently, by clicking **Teaching more!** button many times tenaciously!

If, after some time, by seeing errors which are notified by program, you will see that the error is already relatively not large, and its changes (decreases) are not specially significant as a result of the next steps of the teaching process (compare fig. 5.4) – then you can stop teaching process and try to estimate network knowledge by using exam again.

The results should be better – either when you show some object from teaching data (fig. 5.5), or when you show some object which is not included in teaching data (in other words completely new, unknown by network before), but **similar** to teaching data objects (fig. 5.6).

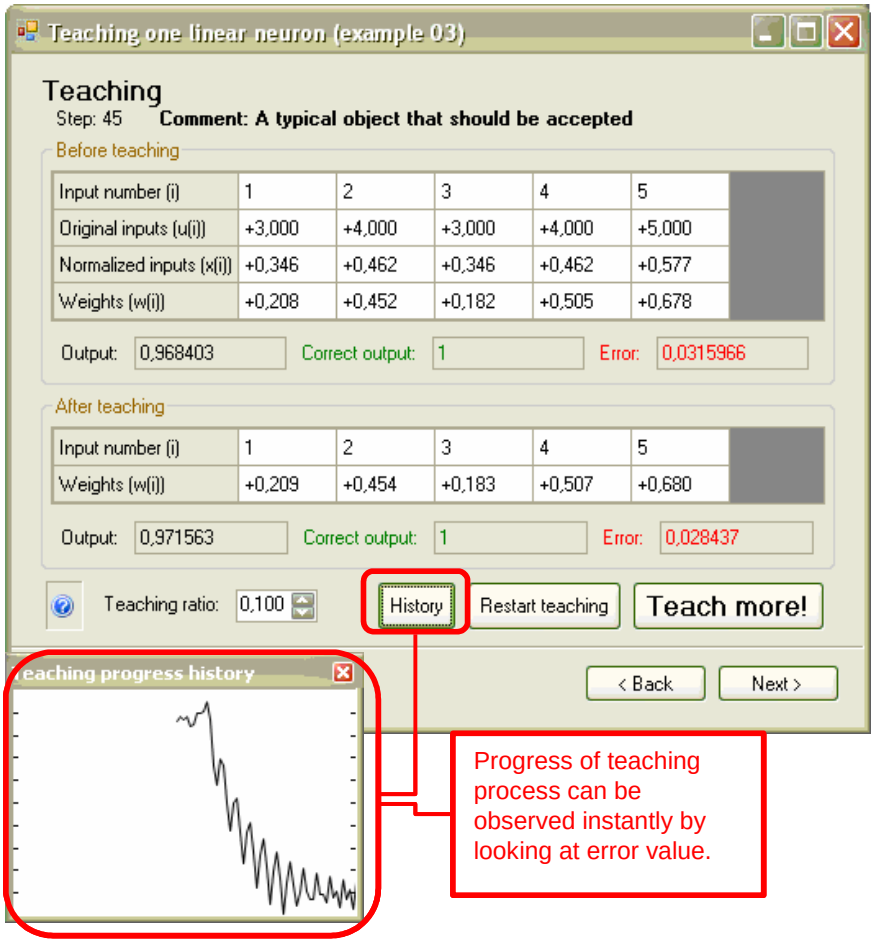


Fig. 5.4. Advanced stage of teaching process is characterized by small value of error in every step and it's not large decrease after teaching

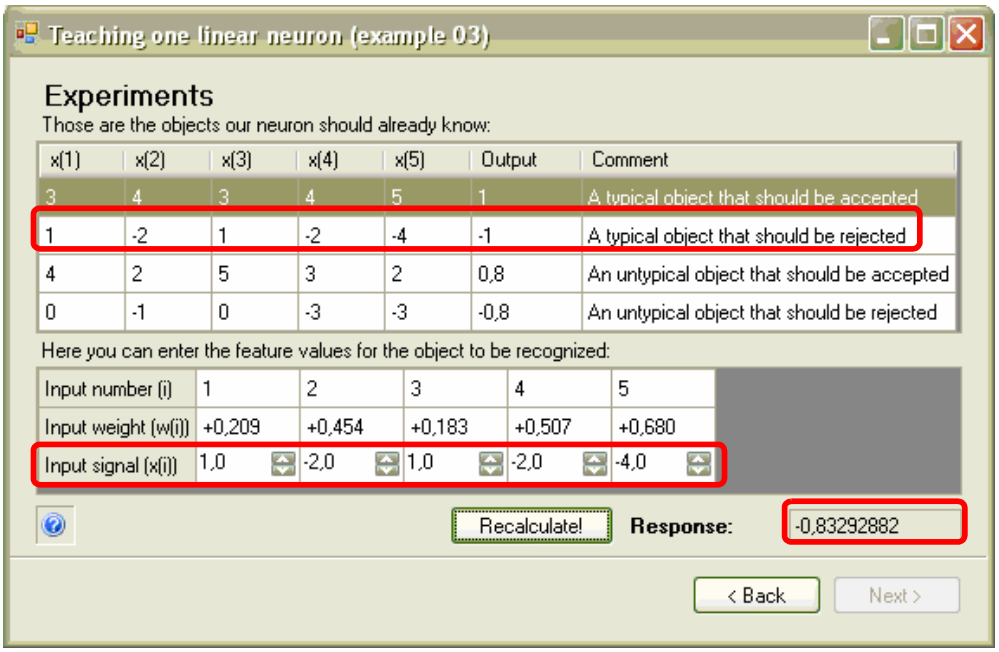


Fig. 5.5. In advanced period of teaching process, the network passes without trouble the examination by rejecting the object which should be rejected

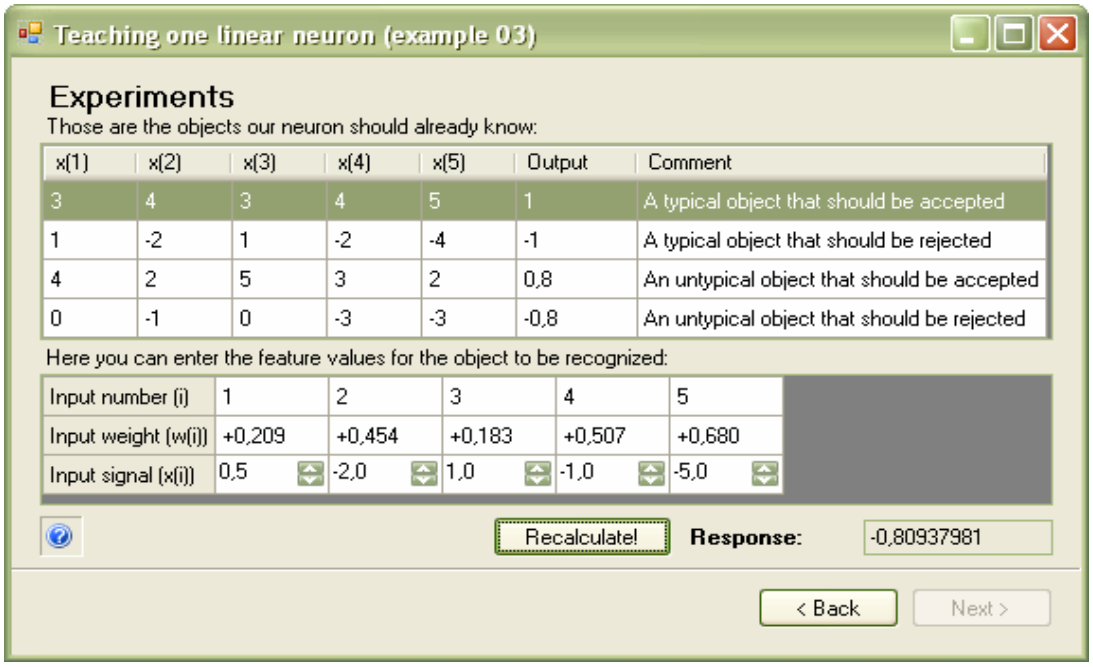


Fig. 5.6. In more advanced period of teaching process, the network shows ability to generalize, because it definitely rejects the object, which is only **similar** to object from teaching set, that should be rejected

During teaching process you can look at the history of this process, which appears (upon request) in the form of changing network error value chart made in the following steps of teaching process.

This useful and instructive chart can be used any time you want on your screen by clicking **History** button.

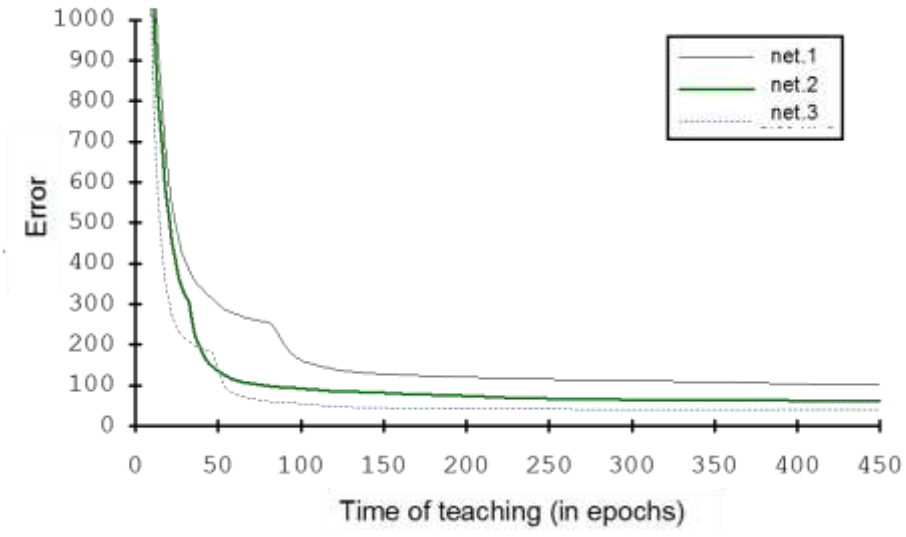


Fig. 5.7. The error decrease during teaching process for different values of initial weights in researched network

Program demonstrates you errors (as primary elements, having influence on teaching course process), and weights value – before and after teaching process correction as well.

Let's notice that by using a small number of teaching data, which in addition are shown to invariably taught neuron (without "shuffling" its sequence, which was advantageous and recommended in one of previous chapters), you cannot demand too much – and certainly you will not achieve a great success in a difficult tasks teaching process. But the simple tasks (the same as task presented in **Default teaching set 03.txt** file) neuron is learning surprisingly fast and effectively.

Bearing in mind(from previous chapters), that correctly trained neuron should create the target standard object itself (in the form of suitable weights values), which is to be recognizable. Then you are able to estimate immediately if the teaching process "follows its nose" (so it may happen sometimes…).

Let's also take into account how dynamic the teaching process follows.

The first few presentations already show significant progress in the range of improving the neuron working correctness, whereas the learning speed decreases after that, but still it is possible to look at decreasing error in the following steps of working networks.

I would like to remind you the figures showing a typical characteristics of one neuron or entire network changing error value in a teaching process (fig. 5.7).

Let's try to check, how would the characteristic of teaching process progress look like in your experiments?

Also let's try to check the fact that you will get **different** characteristics of teaching process in following experiments, despite using the same set of input data. It is the result of the fact that a neuron would start from different randomly drawn initial weight value every time.

Example 03 program is quite "friendly" in use.

With its help, you can freely experiment with a neuron, for example breaking the teaching process up, to examine the neuron and check, how it behaves after using the trial signals – best similar to these by which it was taught, but not the same.

You can come back to the interrupted teaching process every time and "tune up" the neuron a little bit, before it will be examined again.

In this way you can teach the neuron a little bit, then examine it, and teach the rudiments again, and examine again etc.

I only advise you to use successively longer and longer periods of teaching process between the following exams, because changes appearing during teaching process with the passing "epochs" (the following cycles showing all elements of learning data) are less and less noticeable.

It is worth to devote a moment to conduct these types of experiments, because they give very substantial and detailed opinion (all the better, that they base on a simple and easy-to-analyze

example) on one of the most impressive features of neural network– the ability to learn and generalize the achieved knowledge.

5.3. Can neuron have inborn abilities?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

I also advise you to check how "inborn abilities" influence the effect of teaching. To achieve this aim you should repeat the teaching process several times by starting the program from the beginning and observing the occurring changes.

Random process of setting initial weight value will enable you to observe different pace of teaching neuron for the same input data – sometimes neuron learns instantly (it happens that the one is talented from birth!), but sometimes it takes very long to teach one. During the teaching process it happens that neuron has period of time with worse results (error increases in spite of intensive teaching), if the neuron has to overcome some "inborn preferences" during learning process.

This effect of coincidence in the course of the teaching process and its results can sometimes be surprisingly large, what everyone should see in order to believe it.

For that reason do not save time and conduct a few experiments with the use of the suggested program, changing the range in the text of program, in which initial random weight values are set[3].

You can do it by giving initial parameters of weight neuron value other than assumed in *InitializeTeaching () klasy ProgramLogic* method.

For example you can use, instead of instruction

_examinedNeuron.Randomize(_randomGenerator, -0.1, 0.1);

apply the instruction

_examinedNeuron.Randomize(_randomGenerator, -0.4, 0.4);

As a result, initial weights value will have much wider range of changes – and because of that a bigger influence on teaching network process and on its result.

By starting the teaching process several times with the same teaching data, you can easily observe how strong is the influence of the random initial weight value.

The neuron will learn in a completely different way every time!

[3] Unfortunately this experiment requires looking at the program text and changing one of its instructions, and then making a compilation again – it means that this game is intended for more advanced readers of this book.

But I hope that with the use of the previously given tips you will not have a problem with this.

In this way you can tangibly cognize one more property of neuron networks, sometimes demonized (*"free will of an automatic machine!"*) – its **indeterminism**, it means unpredictability of the learning process and its results as well.

Let's take into account the following fact. The network which you use in the experiments described in this chapter, is rather small and not very complicated, and the task which network should perform, is rather easy.

However in the big network and during solving really complicated tasks the addition of many random effects, such as investigated in the described program, can lead to completely unforeseeable network behavior. The effect may sometimes amaze "the scribes" who got used to complete and infallible behavior repetition of regular numeric algorithms, which are used in typical computer programs.

5.4. How strongly neuron should be taught?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

The simple demo-program suggested above can be a "testing ground" on which you should investigate one more important factor, influencing the teaching process, namely the influence of teaching ratio value (describing the pace of teaching) on the characteristic of teaching process.

This factor can be changed in the field **Teaching ratio** in **Teaching** of **Example** window of **03** program.

By giving a larger value of the factor, for example

Teaching ratio = 0,3

instead of used in program

Teaching ratio = 0,1

you can get faster effect of teaching process, but then the characteristic of teaching process will be more "nervous" (with sudden weight value changes and unexpected leaping up and down values of network errors).

Also you can try to use a few different values and precisely observe their influence on the teaching process.

You must remember though that if you give too high value of this factor, the teaching process becomes chaotic and does not give any positive results, because neuron will be "struggling" from one to another extreme, and the final effects will be deplorable– instead of improving its results, neuron will record much more errors.

That effect is also worth to be seen with your own eyes!

On other hand too low teaching ratio value causes that the teaching process will be proceeding very slowly. In practice it means that the teaching progress can be simply unnoticeable and it is highly probable that in practice the neural network user will become discouraged not seeing the progress of its working, and will abandon this method, looking for more effective algorithms.

It is worth interpreting the described experiments with the reference to situations which exist in a real teaching process, in which, instead of artificial neural networks the participants are the real brains of students gaining knowledge.

It is easy to notice that the teaching ratio value expresses "strictness" of a teacher.

Low values of this factor are connected with the situation when the teacher is gentle and lenient – who notices and corrects teaching errors indeed but does not extort the correct answers.

As you have seen on the basis of conducted experiments – such indulgence can lead to poor results.

However too high teaching ratio value, that is too excessive teacher's strictness, may also be harmful.

Very strict punishments falling on an apprentice, determined and hard reprimand after every mistake can lead to frustration shown as the neuron is "struggling" from one extremity to another– without a real progress in the teaching process.

5.5 How to teach a simple network?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

The natural course of events is to proceed from teaching a single neuron to teaching the entire network.

Example 04 program, which has been built by me to reach this aim, is very similar to the program described above designed for one neuron, so it will be easy to use.

You also need file of teaching set data to conduct experiments with this program.

I have prepared and used such file, also calling it **Default teaching set 04.txt**, but this time placing it in such a position that it is connected with **Example 04** program.

The content of that teaching set is quoted below:

5, 3

A typical object that should be accepted by the first neuron

3, 4, 3, 4, 5

1, -1, -1

A typical object that should be accepted by the second neuron

1, -2, 1, -2, -4

-1, 1, -1

A typical object that should be accepted by the third neuron

-3, 2, -5, 3, 1

-1, -1, 1

An untypical object that should be accepted by the first neuron

4, 2, 5, 3, 2

0.8, -1, -1

An untypical object that should be accepted by the second neuron

0, -1, 0, -3, -3

-1, 0.8, -1

An untypical object that should be accepted by the third neuron

-5, 1, -1, 4, 2

-1, -1, 0.8

An untypical object that should be rejected by all neurons

-1, -1, -1, -1, -1

-1, -1, -1

The program, which you will study, informs about the state of teaching process and values of particular variables in a little bit less detailed way than previously described program (**Example 03**). It results from the fact that by using the bigger number of neurons, too precise view into what every single neuron performs can turn out inconvenient – the program would shower you with information. It would be difficult to find this essential information among all of them.

Therefore the insight into the teaching process given by Example 04 program is more synthetic – for each of the three neurons of considered networks instantly, as you can see in the example in figure no 5.8.

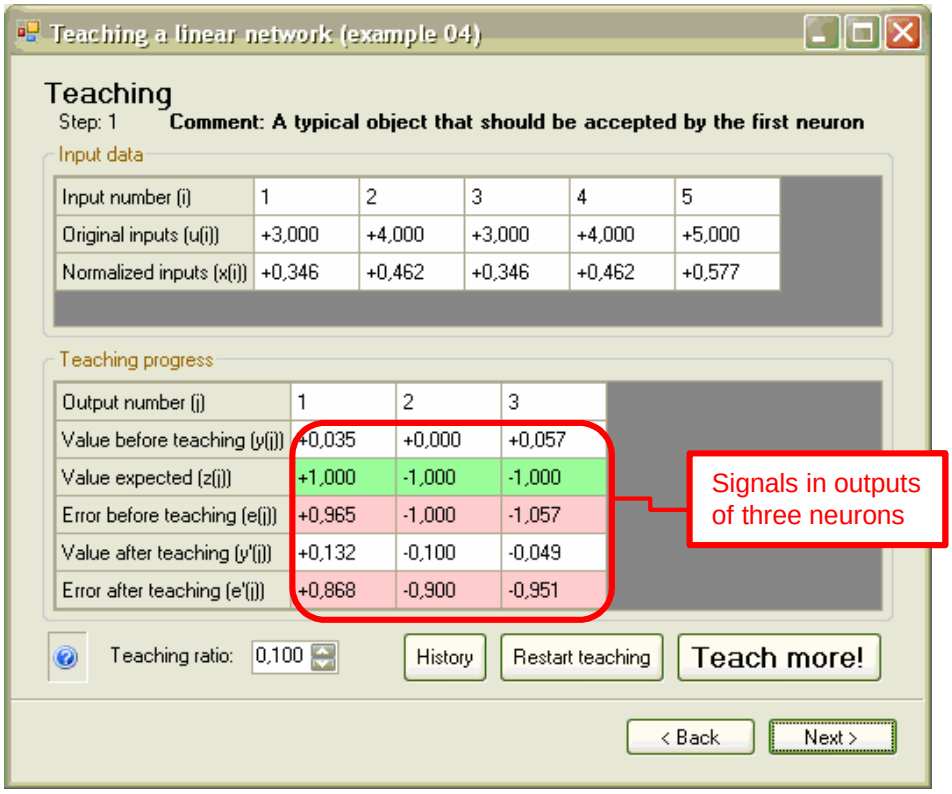


Fig. 5.8. State of network condition at the beginning of teaching process in the **Example 04** program

However it does not interfere with getting a view into the most essential aspect– into the progress of the teaching process, which can be seen by comparing the figure no 5.8 with the figure no 5.9.

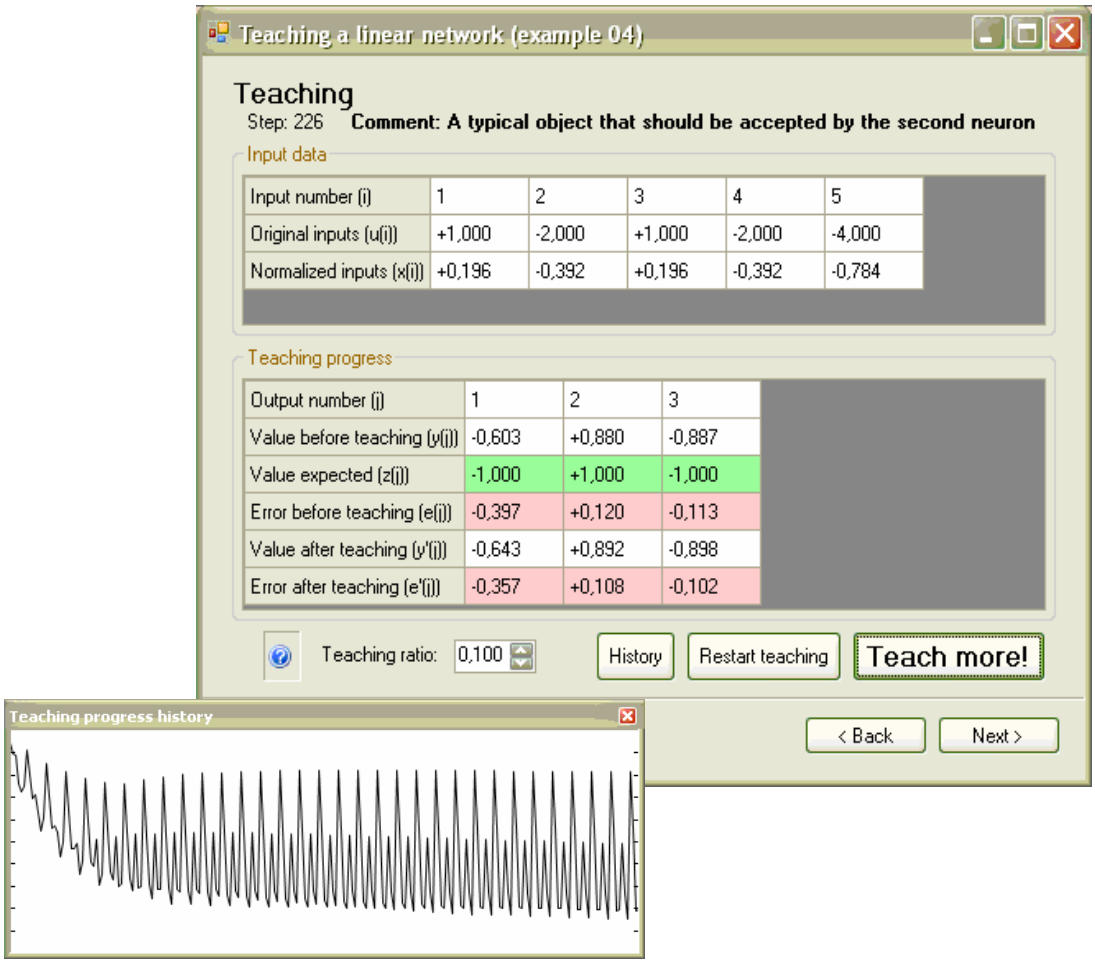


Fig. 5.9. State of network condition at the end of teaching process in the **Example 04** program

The described program, for the same reasons which were mentioned previously, shows network behavior during exam (fig. 5.10) also in a little more synthetic way, which does not disturb the efficient judgment if the network works correctly or not.

As you can see in the figure no 5.10 – the network after teaching process is able to work quite well with the object which has not been shown in the teaching data, but is similar to the object which should have recognized the specific neuron correctly (in this example - the first one).

Let's notice that during exam only the **first** neuron has positive value of output signal, which definitely and unambiguously points to the fact that the recognition process has been finished successfully.

However the ambiguity of this situation causes that neurons no. 2 and 3, which should have definitely renounced this object, have doubts – their answers were hesitant and uncertain.

This situation is quite typical.

Requiring the **generalization** of the knowledge, obtained in the process of teaching, from the network, we can almost always notice its uncertain behavior. It is based on the fact that it is easier to achieve the success (for example correct positive recognition) than to get great **certainty** and

reliability of network actions, in reference to cases that should be definitely and strictly rejected by it.

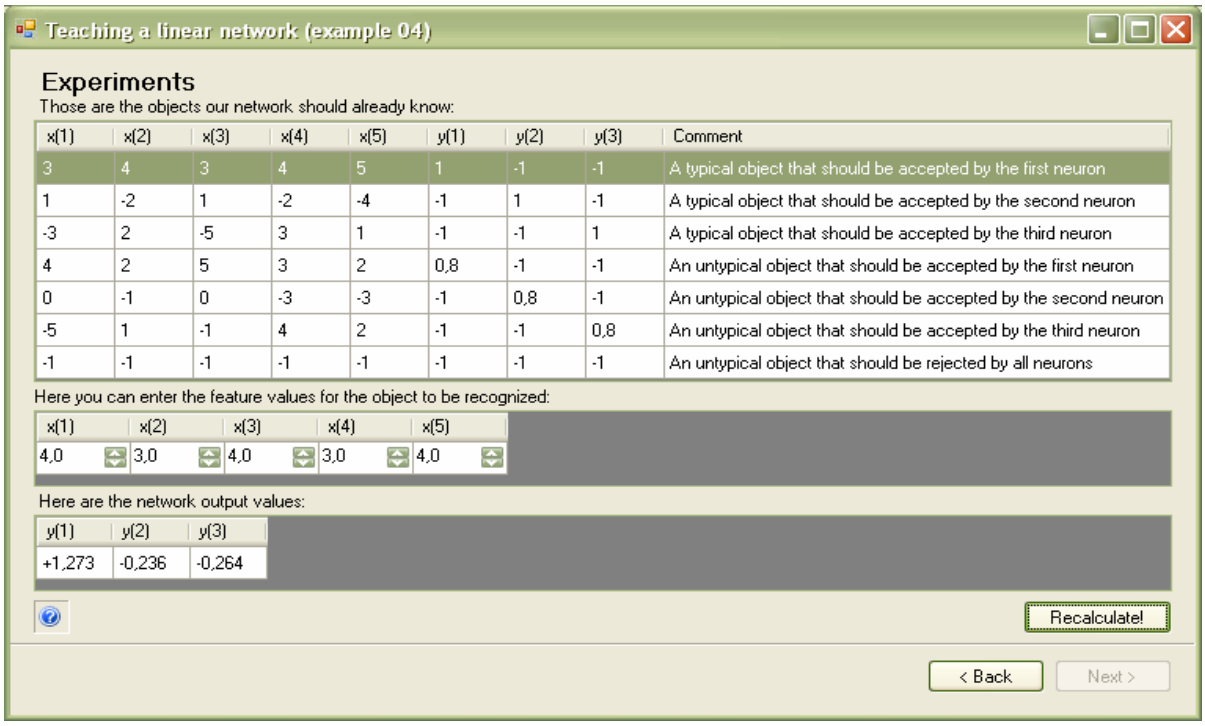


Fig. 5.10. Characteristics of network examination in the **Example 04** program

During the observation of the network teaching process, you can surely notice that among demands which you make during the teaching process (in the form of suitable examples included in a set of teaching data), there are some which are easily and quickly solved by the network, and some other which are much more difficult to solve.

In reference to the previously shown example of the file **Default teaching set 04.txt**, all those tasks are easy, in which there is the necessity of correct recognition of the specific – typical or untypical – target standards by specific neuron. On the contrary, the example of the object which should be rejected by all the neurons, turns out to be difficult to teach. In the figure no. 5.11 I showed one of the very distant stages of the teaching process, where correct recognition of specific objects happens almost correctly, and the object that should be rejected – as you can see in the figure – is still making causing trouble.

Moreover – you can notice that repeated unsuccessful attempts of adapting network to solve this difficult task as well, lead to **deterioration** of up-to-now relatively well skilled network in the scope of solving elementary tasks (which means the typical object recognition).

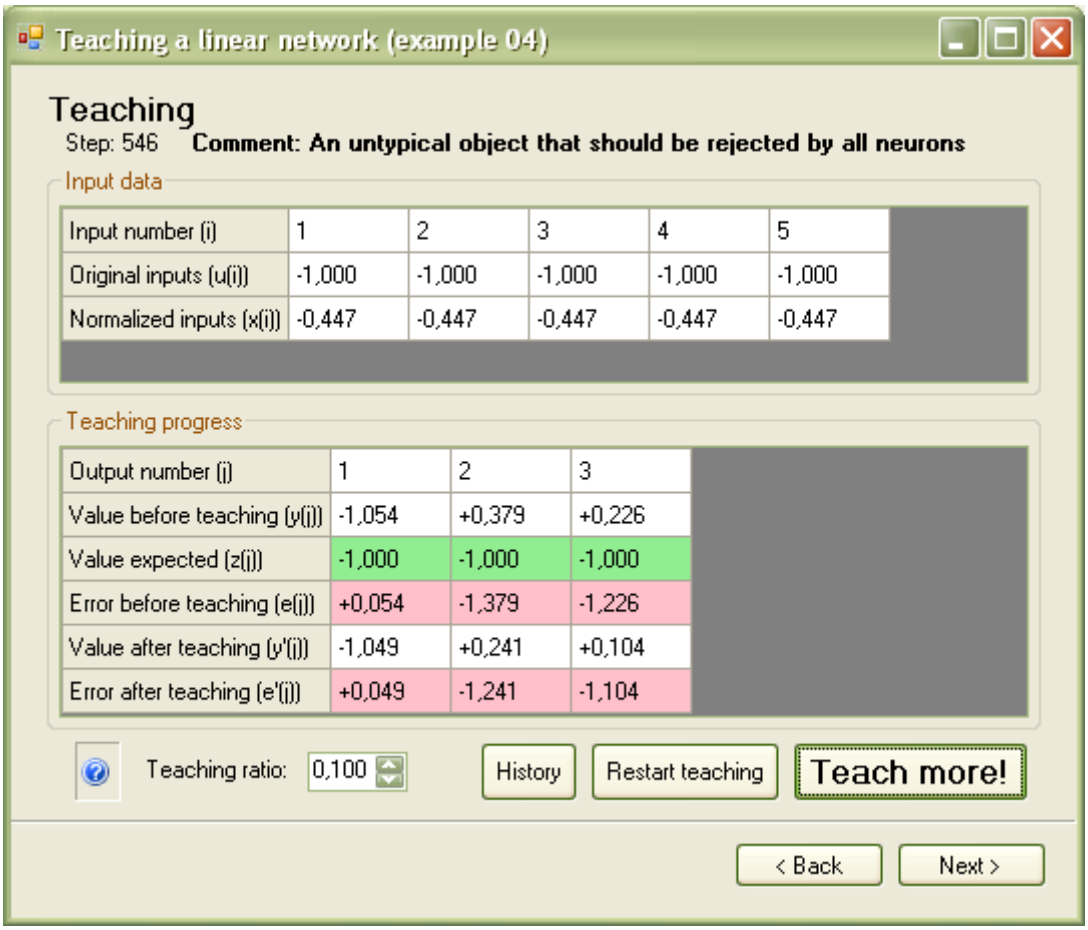


Fig. 5.11. Network teaching in case of recognizing the inconvenient examples

If you encounter this situation in practice – instead of struggling and teaching the network unsuccessfully for a long time –just think if it is possible to reformulate the task being solved in order to prevent such a situation.

Usually the removal of one or few troublesome examples from teaching data radically improves and accelerates the teaching process.

How strongly it improves the skills of learning, you can see for yourself by removing the proper fragment from **Default teaching set 04.txt** file and by teaching the network again – but now without this troublesome element.

You will certainly notice a beneficial change of speed as well as the effectiveness of teaching.

5.6. What are the possibilities of using such simple neural networks?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

I will surely not surprise you (and I also hope that I will not make you sad), if I confess that the network (which you have investigated a moment before) is not the largest and the most complicated neuro-computer that has ever been built in the world.

To tell you the truth, it was so simple that for some of the readers it seemed to be almost primitive.

However even this simple network showed quite interesting forms of behavior and was able to deal neatly with fairly complicated tasks – because collective action of many neurons opens a really wide range of varied possibilities.

Now allow me to make a bit more general remark, namely, it is easy to notice that there are some similarities among single neuron and entire network possibilities.

Therefore, the network can be investigated as the cooperating neurons collectiveness, and the knowledge of single neuron's behavior can be transmitted into the entire network in a quite natural manner.

However there are some uniquely connected possibilities with the network, which a single neuron did not supply.

For example the networks, even so simple, as these described here, enable us to enter boldly in the space of solving **multidimensional** problems, that in other mathematical and computation methods can often cause really serious difficulties. To realize that, enlarge the network dimension in a proper way by assuming the larger value for a number of inputs and outputs to try to use the network for some practical tasks.

The network can be especially used for modeling different complex systems, in which many causes (input signals) influence the occurrence of many results (output signals). You can find out, how many of these applications exist, for example by writing into the Google search engine the headword "*Neural Networks Modeling*" . I have just found the information , that these described examples can be viewed almost 30 million times in the Internet. It would be difficult to find so many practical applications of use for a single neuron!

Apart from creating the neuron models of different complex systems, we can use these simple linear neural networks (like those described here) to adaptable signal processing.

This is again a huge and a very popular discipline – I do not even want to tell you, how many articles Google browser found, when I wrote in the headword "*Signal Processing*".

It is because of the fact that the digital technique created a possibility of computer (and also neuron) processing for many more varied signals– for example speech transmission by phone, pictures recorded by camera or video camera, signals describing patient's body condition, delivered by a modern medical equipment, the results of scientific experiments, usually producing huge quantity of signals, records from control-measuring equipment in industrial automatic systems – and many more.

I think that it would be good to take a short look at this wide, important and interesting discipline, because a signal filtration itself (that is erasing these noises from the signals, which impede perception, analysis and interpretation of data) forms particularly attractive area of implication for neural networks.

5.7. Can network be taught signal filtering?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

Imagine that you have some signal with some noise put on it.

Telecommunication, automatic, electronics and mechatronics engineers torture themselves with such signals every day, so the above situation is not strange.

If you ask one of those experts, what should be done to clear the signal from the noise, then with feeling of superiority you will be instructed that you should use a filter.

If you tackle matter more precisely, you will get to know that a filter is a device (usually – electronic) that allows to pass the useful signal , but stops the noise.

It works very good, and as the result of that fact we have efficiently working phones, radios, TV-sets etc.

But each expert will confirm that it is possible to create a good filter only when the noise has some property which is not present in the useful signal.

If something has this property – than it should be stopped by the filter as a noise.

If it does not have this property – than it is allowed to pass.

It is simple and effective!

However, to get the system work you have to know a lot about a noise, that disturbs your signal.

Without the needed knowledge you will not create a filter, because the filter does not possess the data, which information should be kept and which should be stopped.

Unfortunately, it often happens that you do not know what the source of the noise disturbing your signal is, and what its properties are.

If you send a probe rocket somewhere far away into space with the intention to gather signals describing for example an unknown planetoid – then you do not know, what rubbish may come and hang about with signals from the probe during its journey over millions of kilometers in interplanetary space.

How to create a filter then?!

Yet, it is possible to separate a useful signal from **unknown** noise by using adaptive filtration method.

In this case adaptive means teaching the signal receiving device the previously unknown rules of separating the signals from noise. Particularly, the neural networks can be trained to filter and select signal from noise.

To achieve this, samples of disturbed signal are treated as input signals for network, and “clean” signals are used as output signals.

After some time the network will learn to select the undisturbed output signals from disturbed ones and will be able to work as a filter.

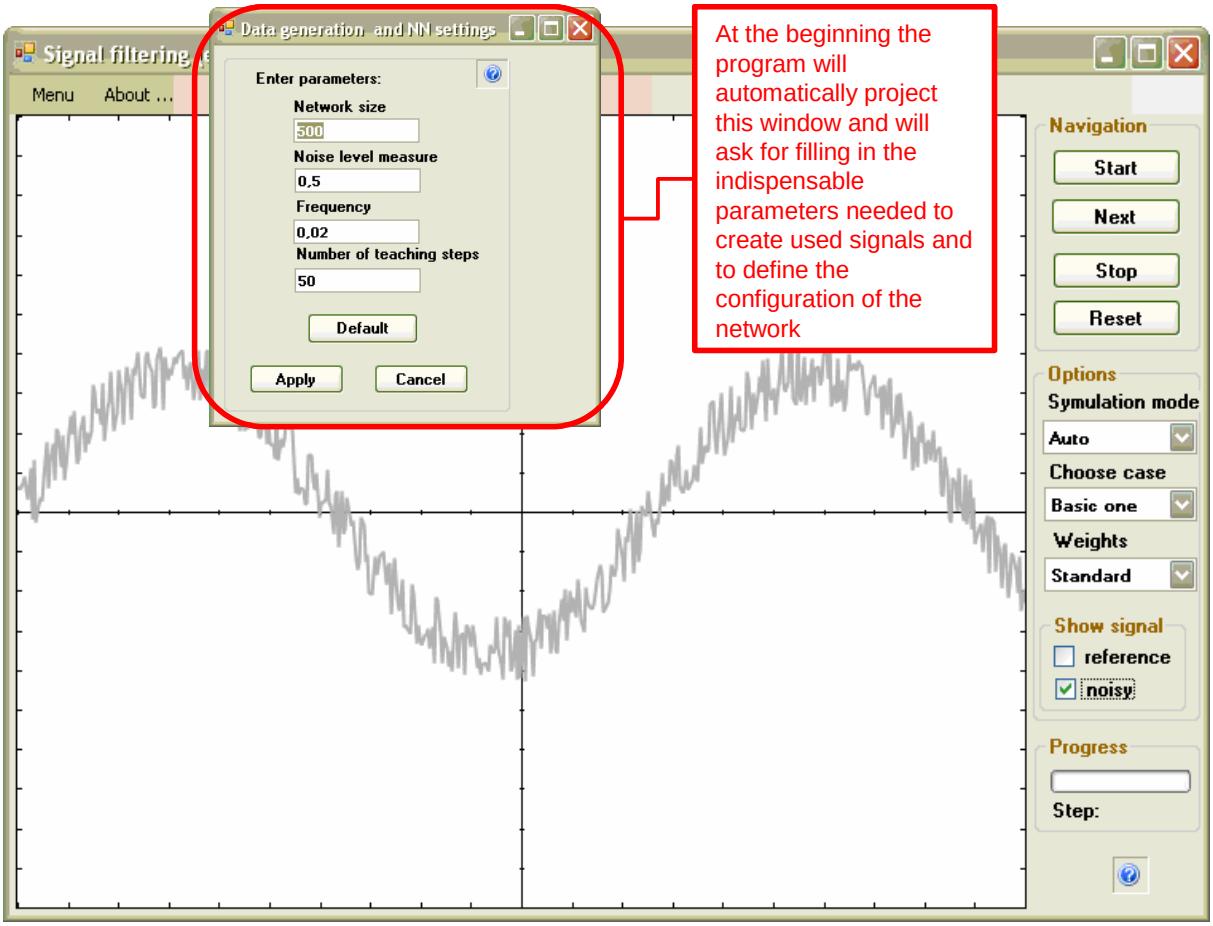


Fig. 5.12. Disturbed signal which has to be filtered by neural network

Let's consider a real example.

Let's take a standard signal – for example a part of sinusoid signal – both a "clean" and a disturbed one.

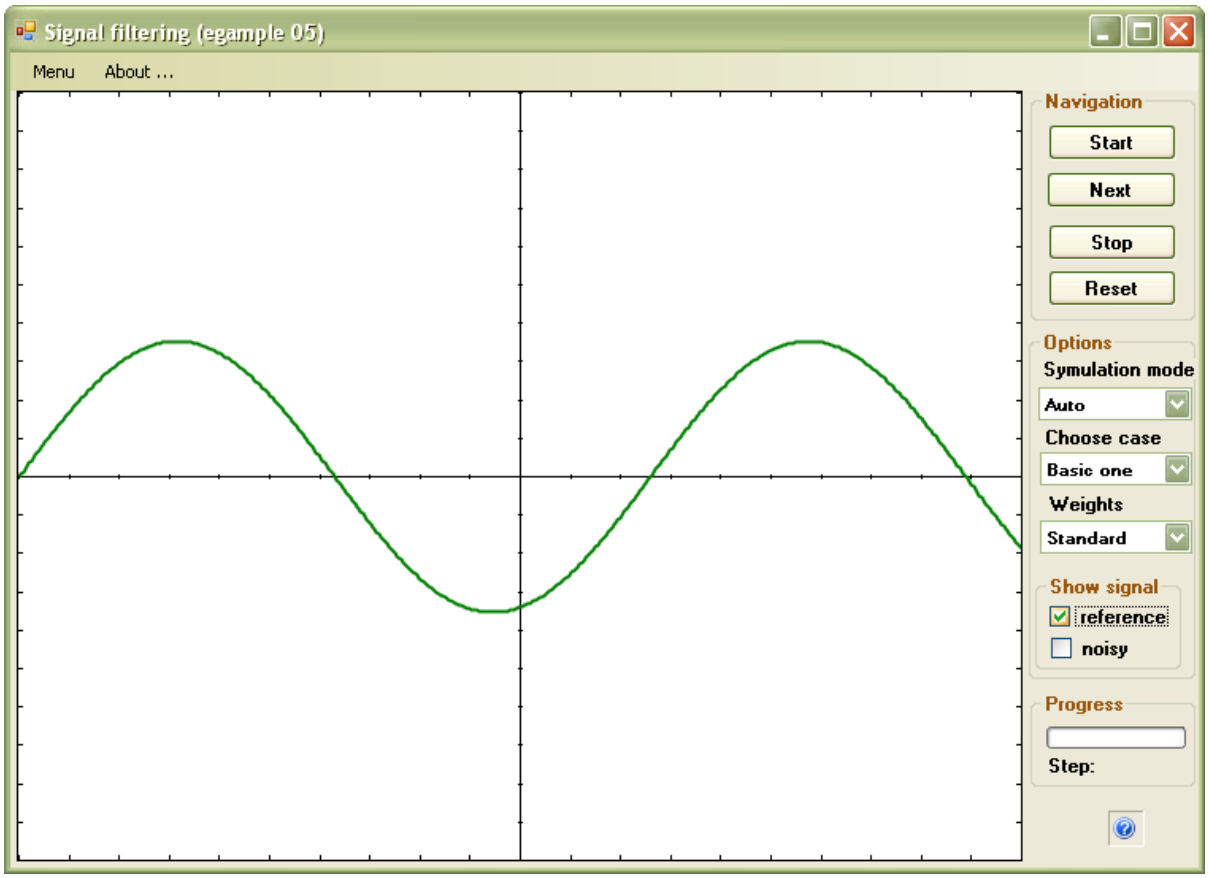


Fig. 5.13. Standard signal which has to be reconstructed by neural network

These signals – in the form of a file which allows to teach networks– can be created by **Example 05** program.

This program will automatically produce a file with data called **teaching_set**, which can be used to teach a simple network.

The window in which you can modify the required parameters to generate this file, appears when we start the **Example 05** program.

This window is shown in the figure 5.12.

Unfortunately, in this window the program demands from you to give the network size (**Network size**), expected noise value (**Noise level measure**), frequency (**Frequency**) and number of teaching steps (**Number of teaching steps**).

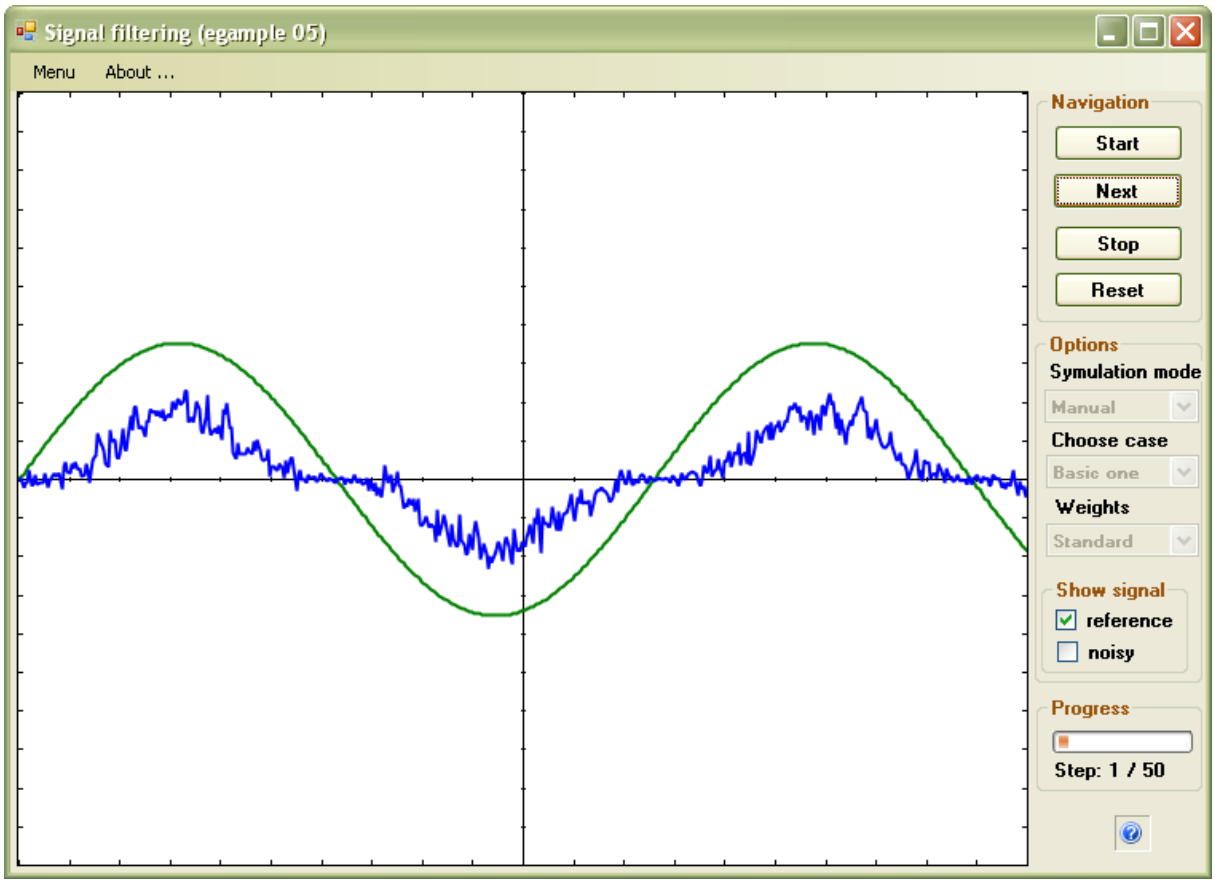


Fig. 5.14. Results of signal filtering after one step of teaching process

Looking at all these needed details, you can definitely become discouraged from the fun of working with this program, because it looks mysteriously and vague, and in addition where would you know these all necessary values from? It is not as bad as it seems, though.

Looking closely at the described window, you will find some information written in each of the windows.

These values are selected and tested by me for the program to work well and show interesting effects. Therefore if you do not have better ideas, then at the beginning you can use these default settings and simply (without further ado) accept it by clicking the button **Apply**.

Of course in future, when you have the will and ability, each of these values can be changed at will – at least just to observe “what happens, if”.

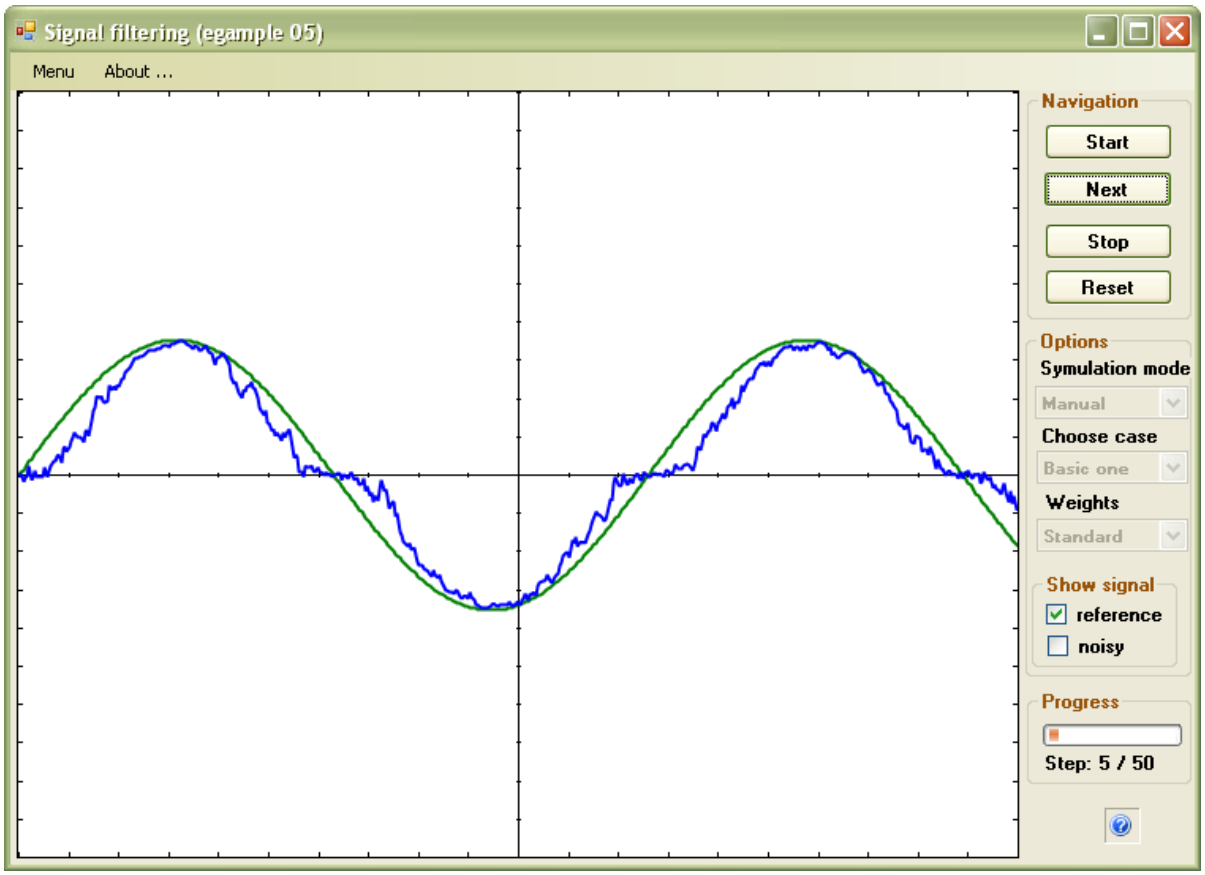


Fig. 5.15. Results of signal filtering after five steps of teaching process

The freedom, that the program gives you in this field, is very large.

The values that have been already introduced, can be erased any time (**Cancel button**), and you can also immediately reconstruct again all the settings designed by me using **Default** button. As the result the **Example 05** program can really be liked, , despite the unfriendly impression you can have at the beginning, and by using its operations, you can observe the work of the network, which will learn to filter the signals by itself.

During the experiment the network will be correcting signal step by step, but every time you can observe what the not-filtered signal (fig. 5.12) and the original signal, that is the signal without noise (fig. 5.13) look like.

To achieve this aim, it is enough to mark proper option (**reference** or **noisy**) in **Show signal** group in the range of the right margin displayed by the program on the screen.

Now, let's take a look at how it works.

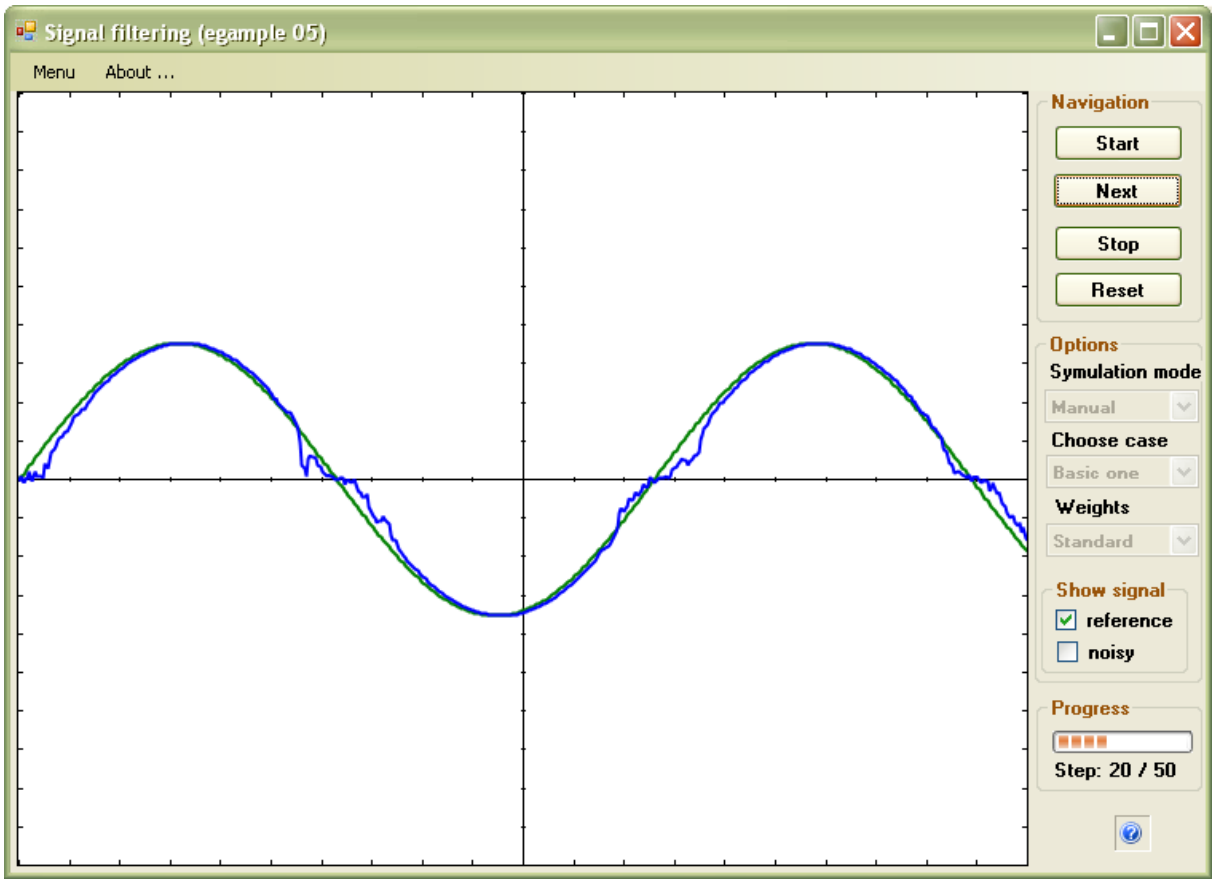


Fig. 5.16. Results of signal filtering after twenty steps of teaching process

The first results of filtration, after few steps, are not very promising (fig. 5.14 and 5.15), but the constant teaching of the network (by repeated clicking the button **Next**) leads to the fact that finally the well trained network learns to filter signals in a nearly perfect way (fig. 5.16).

The teaching process can proceed automatically or step by step.

It means that the program shows, at your request, all its mysteries or can pass many stages of teaching automatically, during which the network makes its work perfect, and you can view only the final effect at ease.

You decide about the teaching mode of the process, depending on your choice of **Simulation mode** group(**Auto** or **Manual**).

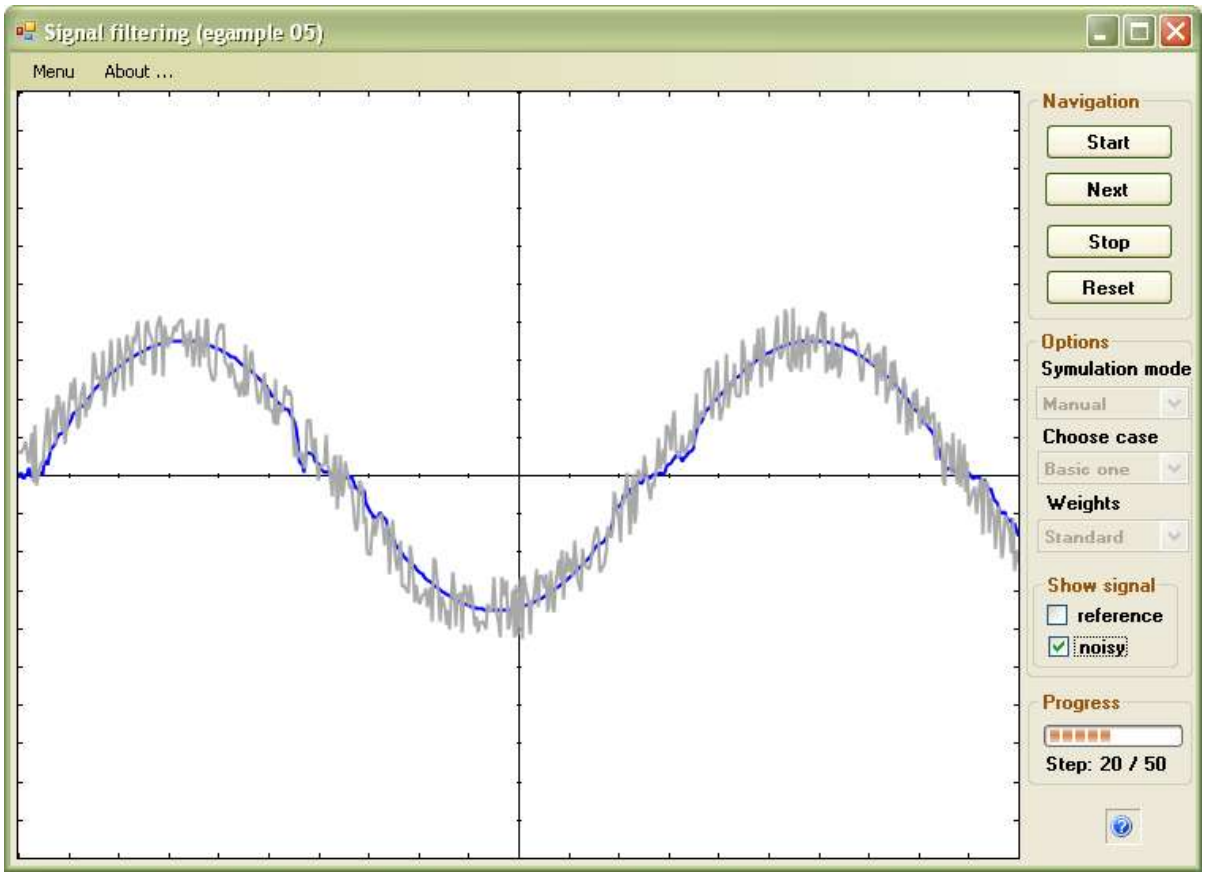


Fig. 5.17. Estimation of effectiveness of signal filtration after twenty steps of teaching

At the beginning, during first experiments, it is worth to observe step by step how the network teaching process proceeds, and if it is the case, one should choose **Manual** mode.

During next experiments it is possible to modify the number of steps (by using **Menu->Configuration** option on the upper edge of our program's window) and try the longer teaching – for example 50 or 100 steps using **Auto** mode.

The results are very informative, so it is worth to make an effort!

As you can see in the presented examples – the network really learns and improves its work in such a way, that after some time (pretty short!), it quite effectively erases accidental disturbances appearing in the original signal.

Filter effectiveness, which is generated as the result of using the network teaching process, can be estimated by putting the picture of the signal before filtering on the process of the filtrated signal (fig. 5.17).

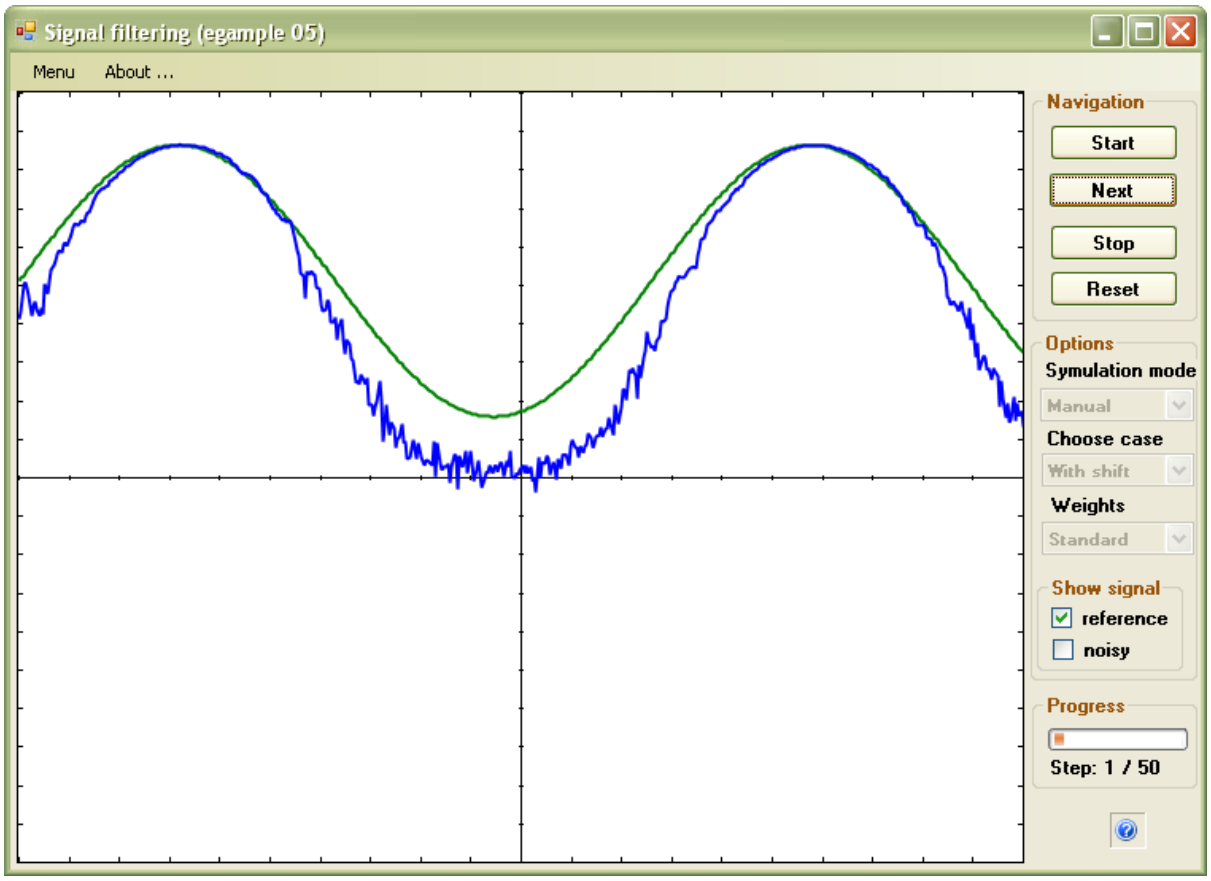


Fig. 5.18. Results of filtration of a moved signal after one step of teaching

While experimenting with the program you will see that the network can really learn to filter the signal and after some time performs this task quite well.

It is also easy to notice that the most difficult task is to teach the network to reproduce signal in a place, where the values of the standard signal are small (particularly if they equal zero), therefore in the **Example 05** program it is possible to use two versions of network teaching – first with the original sinusoid signal, and after that by sinusoid signal moved in such a way that the values processed by the network could be only positive.

You choose the option with the window "choose case" (**Choose case** on the right side).

At the beginning, the results in the second case (definite in **Choose case** window as "with displacement" – **With shift**) seem to be worse (see fig. 5.18), however persistent network teaching, in this case, gives much better results than the mentioned before (fig. 5.19).

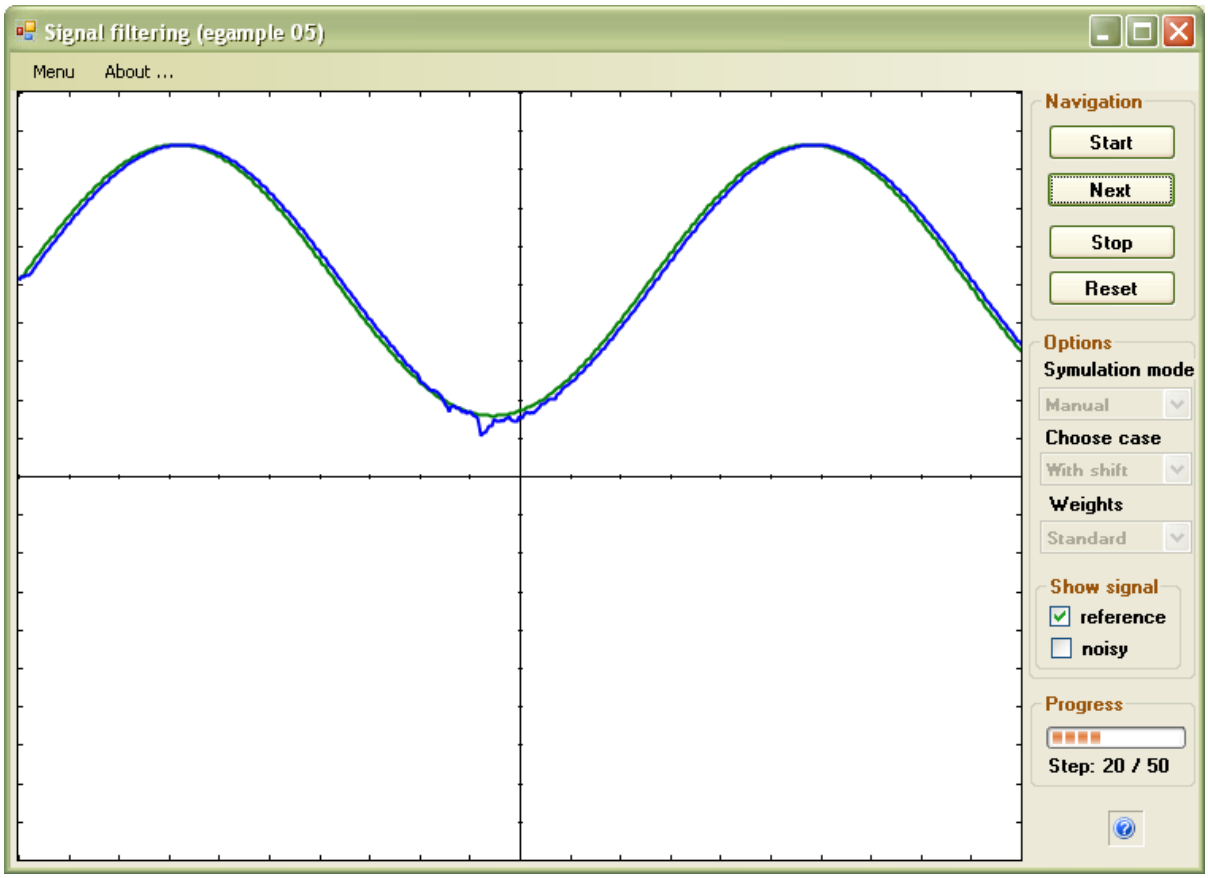


Fig. 5.19. Results of filtration of a moved signal after twenty steps of teaching

The observation of the network behavior in both considered variants will help you in the future to detect and analyze the causes of possible failures in the process of using the network to much more complicated tasks.

I hope that the teaching process of the network filtering the signals was interesting for you and helped you to understand how and in what way the network performs tasks during its work improvement.

However, taking into account the entire attractiveness of tasks which were solved in this chapter by us I have to confess that the linear networks are just 'the kindergarten' of the neural networks, they are some kind of warm up.

These networks can only be single layer (if you want to know why – look at the end of chapter 3 of my book entitled ***Neural Networks***, it is not possible to explain that without mathematics, but I have promised not to use it here), meanwhile the cerebral cortex is MULTILAYER...

So, the real adventure will really begin when you create and activate your first multilayer network created from nonlinear neurons.

As you can guess yourself, it will happen quite soon – in the next chapter…

5.8. Questions and tasks to individual solution

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

1. Why during network teaching process of much complicated tasks do we use teaching set saved on a disc, and we do not teach networks by giving the proper data to the program using a mouse or keyboard?

2. What is the reason why the chart in figure 5.4 (and many other charts of error changes, which you will observe during teaching process) apart from unlucky declining tendency, that determines the progress of teaching and systematic error reduction, has "protuberances" upstairs? Does the neuron suddenly 'go crazy' from time to time?

3. Try to estimate (on the basis of experiments described in this chapter) to what degree the teaching process, in the form of a particular behavior, has the influence on the final result, and on the contrary, to what degree " the inborn abilities" which result from a random initialization of the parameters ranging in the network influence its work?

   Can you draw some practical conclusions about the influence of your education on your life career?

4. Try to establish by experiments (starting many times Example 03 program), what the most profitable value of teaching ratio (*Teaching ratio*) is in the task that was solved by the above described network. Will, in your opinion, in another task, described by completely different teaching set, the optimal value of teaching ratio be the same – or maybe other?

5. Think when you can use bigger values of teaching ratio (*Teaching ratio*): in the tasks that are easy to solve by the neural network, or in the tasks that are difficult and complicated?

6. Try to invent and use another teaching set cooperating with Example 04 program, with the use of editor Notepad of Windows system or (better) with the use of specialized tool that is Visual Studio, then save the new content in Default teaching set 04.txt file.

   Try to come to the point when Example 04 program becomes a universal tool for you, which enables you the solving of different tasks, depending on a teaching set, randomly selected by you.

7. Prepare a few files with different teaching sets and try to compare, how well the network can be taught, in case of each different task.

   Establish, what degree of task difficulty (measured by the degree of similarity of data sets describing these objects – especially atypical – which have to be distinguished by the network) influences the time of teaching and level of network errors, after finished teaching process. Try to find such a difficult task, that the network will be unable to learn, no matter how long the teaching process is.

8. Examine, how the teaching process of Example 04 program proceeds in relation to initial weight value, teaching ratio and different modifications of teaching set.

9. The neural network that learns the adaptive signal filtration has at its disposal: the signal disturbed by noise and the standard of undisturbed signal.

   Why isn't only the disturbed signal(noise) used as a model during network teaching?

10. The figures no 5.18 and 5.19 show, that the filtration conducted by Example 05 program is better in the upper part of chart than in the lower. Why?

11. Task for advanced students:

    Example 03 program used the teaching neural network to create a classifier (a network, which after being given a specified set of inputs signals would produce a signal on output, which could be interpreted as an acceptation of object described by data or absence of such acceptation).

    Examine how this program will behave, when it is forced to learn a more difficult trick: calculation on the basis of inputs data value, the definite output value.

    This network will be able to work as model of some simple physical or economical phenomenon.

    That is example data set in the form of Default teaching set.txt, for which this model can be created (only two inputs for the network were assumed, in order not to use too much teaching data):

    2, 1

    Observation 1

    3, 4,

    -0.1

    Observation 2

    1, -2

    0.7

    Observation 3

    4, 2

    -0.2

    Observation 4

    0, -1

0.3

Observation 5

4, -5

1.9

Observation 6

-3, -3

0.6

Observation 7

-2, -4

1

Observation 8

3, -2

0.9

Observation 9

-1, -1

0.2

Make sure, that the program gives solutions for another (not used during teaching) data set correctly, knowing that the teaching data were obtained from the problem described with equation $y=0,1X_1-0,3X_2$.

12. Task for advanced students:

    Example 05 program is a program that illustrates the work of adaptive filter based on the neural network being taught, and not working program, which is designed for practical uses.

    However the same network can be used to filter other signals – for example ECG record.

    This signal in digital form is surely not at your disposal, but try to use proper modified network for others signals – for example filter a sample of sound in the form of WAV or MP3 file. And if you like particularly difficult tasks then think about how similar rule of adaptive filter network can be used for picture processing?

6. Nonlinear networks

6.1. Why do we need non-linearity?

(Translation by Weronika Łabaj, weronika.labaj@googlemail.com)

Linear systems (not only neural networks but all linear systems in general) have number of nice qualities. Their behaviour is well-known and predictable, mathematical description is simple and we are always able to find a solution. You might ask what's the point of leaving that comfortable, cosy room and replace it with complex and difficult non-linear networks. Does it make any sense?

Well, we have to do it, as it happens.

- Why?

There are number of reasons (you will learn about them later in that chapter), but for the time being let me only say this: **Using network created from non-linear neurons we can solve many more types of problems than when using the linear ones.** With linear network we can only find solutions of one class of problems - where the correlation between output and input signals is linear. Non-linear networks are much more flexible.

If you are fluent in math it should be clear by now what the problem is. But if math is not your strong suit don't worry. In that book you won't find any single mathematical formula. I'm not going to leave you alone right now in fear that above explanation wasn't clear enough. Is there anything extraordinary in a fact that network created from linear neurons deals only with linear functions? At first it might seem to be a bit confusing.

If you were fluent in math I would only say (and that explanation should do) that in case of linear transformation we can use **transformation matrix** to get the output of the network being given only the input signal. Output signals are vectors in that case. Well great, but for you **matrix** is probably a Hollywood blockbuster, so I'm afraid that my explanation might be not clear enough yet.

So I'm going to compare **linear** and **non-linear transformations** by showing their basic properties, that is **homogeneity** and **additivity** (also called **superposition property**). You don't have to remember those complicated words, it doesn't matter. What is important is getting the idea of what they stand for.

Homogeneity means that if the argument is multiplied by a factor, then the result is multiplied by some power of this factor. You can think of it as a **cause (or trigger)** and **result**. It doesn't matter what the cause and result represent in the real world. Just remember that if you know the **cause** you can predict the **result** because you're familiar with the **rule, the correlation** between them. **Multiplicative** scaling behavior isn't the only possible linear correlation though. Quite contrary, in the real world you can find many examples of correlations in which you don't observe multiplicative scaling. For example you know that the more you study the better your grades are. So one can say there's a direct correlation between your efforts and grades you get. We can present it as a kind of transformation. However **it's not true** to say that if you learn twice more than your grade will be

twice better. So the correlation between your efforts and grade is **non-linear** one because it's not homogenous.

The other requirement of linearity is additivity. It's also quite easy despite its complicated name. When investigating some kind of transformation you check how does it behave after triggering it and then what is the result when the cause (trigger) is modified. You know the end result for the first trigger and for the second one – then we try to predict what will be the result if both of them work simultaneously. If the result is the exact **sum** of the results we got using triggers separately then the phenomena (and therefore the transformation or function we study) is **additive.** Unfortunately we don't know about many processes and phenomena that are **additive** - they simply aren't common in nature. If you fill up a pitcher with water then you can put flowers in it. If it falls from the table – it's going to lie there. But if you fill up a pitcher with water and it falls from the table you're not able to put flowers in it. A pitcher full of water falling from the table is going to break for sure. The fact of breaking the pitcher is therefore non-linear phenomena and can't be predicted as a simple sum of the activities that were studied separately.

While creating mathematical descriptions (models) of systems, processes or phenomena we prefer linear functions as they are actually easier to use. If we want to use them in neural network it's really great, as it's going to be probably nice and simple network.

Unfortunately, many systems, processes or phenomena can't be put into "frames" of linearity. We have to use complex non-linear mathematical models for them –or – what is easier and more comfortable – we can create non-linear neural networks that model those systems, processes or phenomena. In that case neural network is extremely powerful and efficient tool. In a huge, multilevel linear network the correlation between the input and output can be actually anything. This fact can be derived from one of the basic mathematical theorems about function interpolation and extrapolation, linked with a great Russian mathematician - Kolmogorov.

It isn't the right place to go into details in here, the problem is highly theoretical and mathematical. So for the time being it's enough to simply trust me. Let me only assure you that superiority of non-linear networks over linear ones isn't only academic problem (such as "scientific" discussions *on Easter being better than Christmas*) but is an extremely practical issue. There are quite a lot of problems, very **practical** problems that can be solved **only** with non-linear networks, linear ones wouldn't do for them. One of the programs you're going to study in this chapter is going to make it clearer but for now try playing with some other programs showing how the non-linear networks work and what are their capacities.

6.2. How does nonlinear neuron work?

Not translated yet …

6.3. How does work network made from nonlinear neurons?

(Translated by Michał Majewicz, mmajewicz@gmail.com)

In the above described program, you used a neuron, which output accepted two values, which could be associated with the acceptance (recognition) of certain set of signals or with its total rejection.

This acceptance or rejection is often connected with **recognition** of an object or a situation, that is why networks built with those neurons, which you examined in the program **Example 06a**, are often called **perceptrons**.

Perceptron types of neurons generate on their outputs only two types of signals: **1** or **0**.

It represents the rule (in limited mark) known from neurophysiology as "Everything or nothing", referring to the typical act of a real biological neuron. This is how the above described program **Example 06a** acts and with such neurons you will build all consecutive, described in this books networks. Now you will trace a way of acting and learning of a group of bipolar neurons by studying an example of simple one layered, nonlinear network, built with this type of neurons. Such network is described in the next program **Example 06b**. For teaching network from this program, you can use the file **Default teaching set 06b.txt**, defaulted by **Example 06b** and containing examples, witch which you will teach your network what to do.

This file may have for example the following structure:

5, 3

A typical object that should be recognized by the first neuron

3, 4, 3, 4, 5

1, -1, -1

A typical object that should be recognized by the second neuron

1, -2, 1, -2, -4

-1, 1, -1

A typical object that should be recognized by the third neuron

-3, 2, -5, 3, 1

-1, -1, 1

Content of a similar file has been earlier discussed, so it seems, that there is no need to repeat a detailed discussion of its content – however if you have some doubts at this point, look up the previous chapter.

Comparing presented above file to the one you have used earlier, notice, that it suffices a small number of examples, because – as you will find as soon as you start the program – the nonlinear

network can learn very quickly. Usually[4] suffices just one step of learning to eliminate appearing errors (Fig. 6.5).

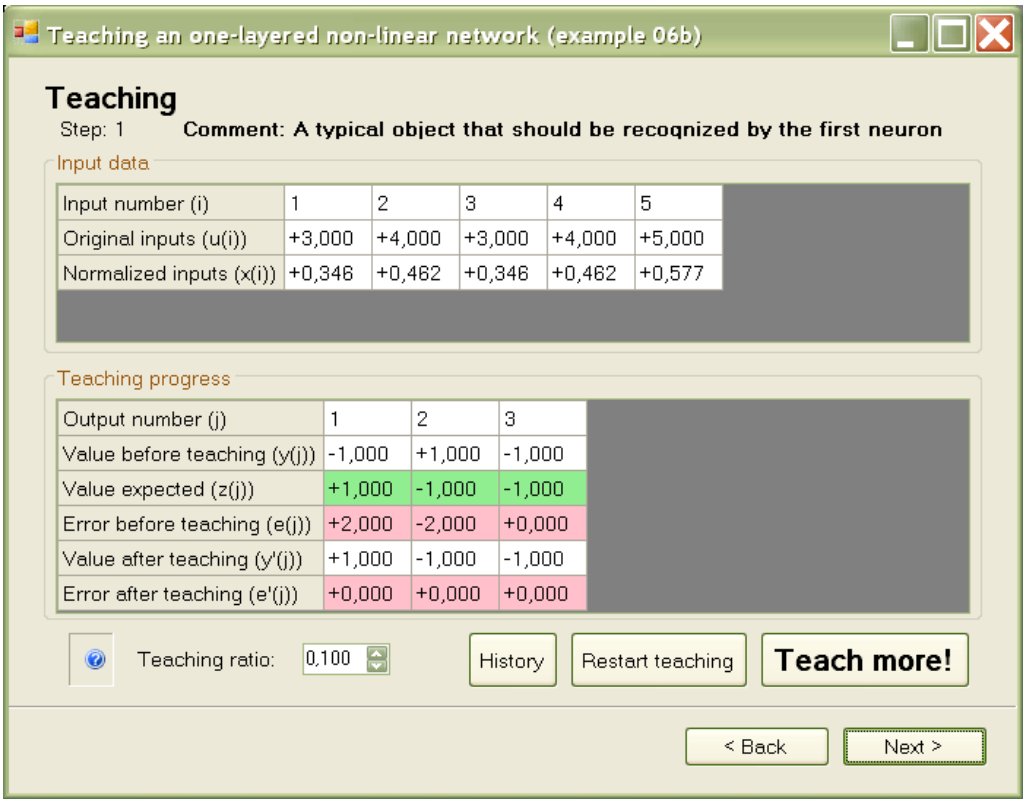


Fig. 6.5 Just in the first step of teaching of nonlinear network one succeeds to remove majority of the errors

On average the network is totally taught after 6 - 7 steps and then one can conduct the examination. An exam evidences, that network not only quickly learns, but also nicely generalizes the captured knowledge (Fig. 6.6).

[4] Conditional form of this statement results from the fact, that in neuronal networks, one can never be completely sure of everything - each try, each step of the teaching process may be different, because random factors have strong influence on the final result. These factors are never identical twice. Completely as in life, as in **your** life!

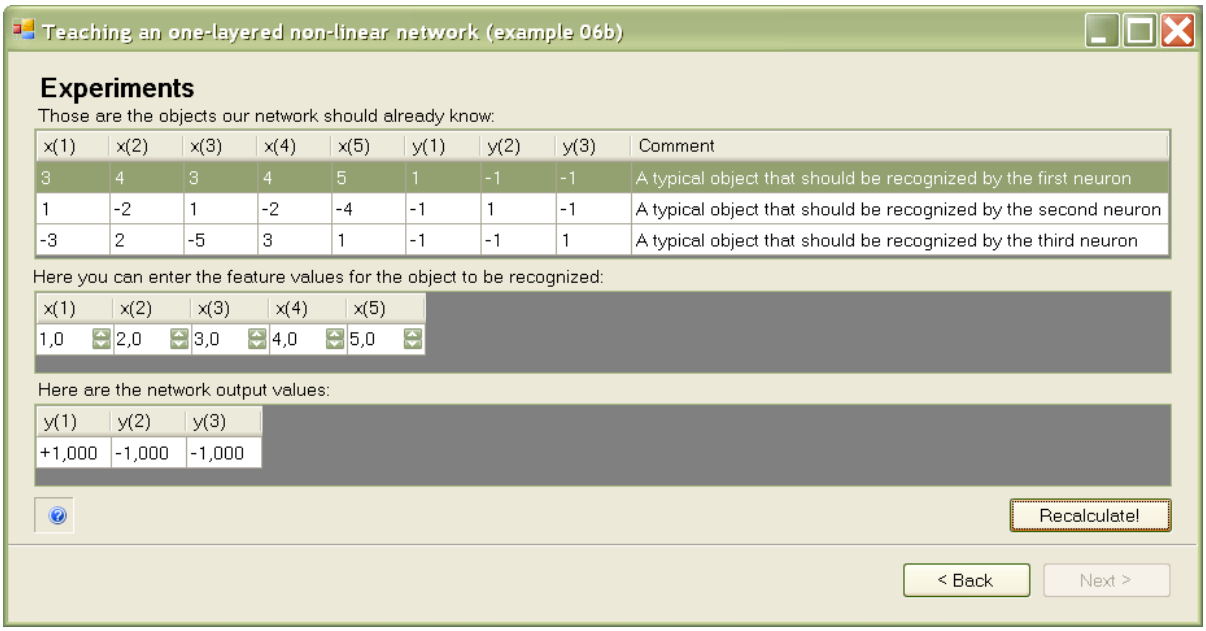


Fig. 6.6 Nonlinear network well generalizes knowledge, gained during teaching

The effect of the knowledge generalization in the neuronal networks has always been and is very important. What would be the benefit if you taught a neuronal network a series of tasks with the correct solutions when you would later recognized that network cannot solve any other task but only those that you taught it earlier? After all there is no need to solve tasks from teaching set, because solutions for these cases are already known. What you really need is a tool, which first learns solving tasks from a teaching set and later can also solve **other tasks**. Fortunately neuronal network can **generalize** knowledge, by also skillfully solving these tasks, which are not included in the teaching set, but in a sense are **similar** to those ***teaching tasks*** (they are based on the similar logic of the relationships between input and output).

The property of the generalizing of knowledge is one of the most important features of neuronal network, that is why I encourage you to check how far can you disturb the input data in relation to the data on which the network has been taught, so the results generated by the network were still sensible and could be recognized as the effect of generalizing gained knowledge, but not entirely free fantasizing – of which the network is also capable! I warmly recommend that you try yourself again, to "bully" the network in several other ways – and I am sure that you would adjudge its superiority over linear networks!

6.4. How to present action of nonlinear neurons?

Not translated yet …

6.5. What are the capabilities of multilayer network of nonlinear neurons?

Not translated yet …

6.6. How the learning of nonlinear neuron proceeds?

Not translated yet ...

6.7. Which research can be performed during the neuron learning?

Not translated yet ...

6.8. Questions and tasks to individual solution

Not translated yet ...

7. Backpropagation

7.1. What is backpropagation?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

In the previous chapter we were discussing some chosen aspects of functioning and teaching a **single-layer** neural network built from non-linear elements. Here, we continue the analysis showing how the **multi-layer** non-linear networks can work. Such networks have, as you already know, more significant and interesting possibilities – which you may have found checking the functioning of the program **Example 06.** Today we will talk about how such networks can be used and taught.

You have already got the information that we are going to build multi-layer networks from non-linear neurons. You also know how a non-linear neuron can be taught (revise the program **Example 06)**. However, you do not know (and even if you know you have not felt it!) that the basic problem with teaching such multi-layer neural networks built out of non-linear neurons is the problem of so-called **hidden layers** (fig. 7.1).

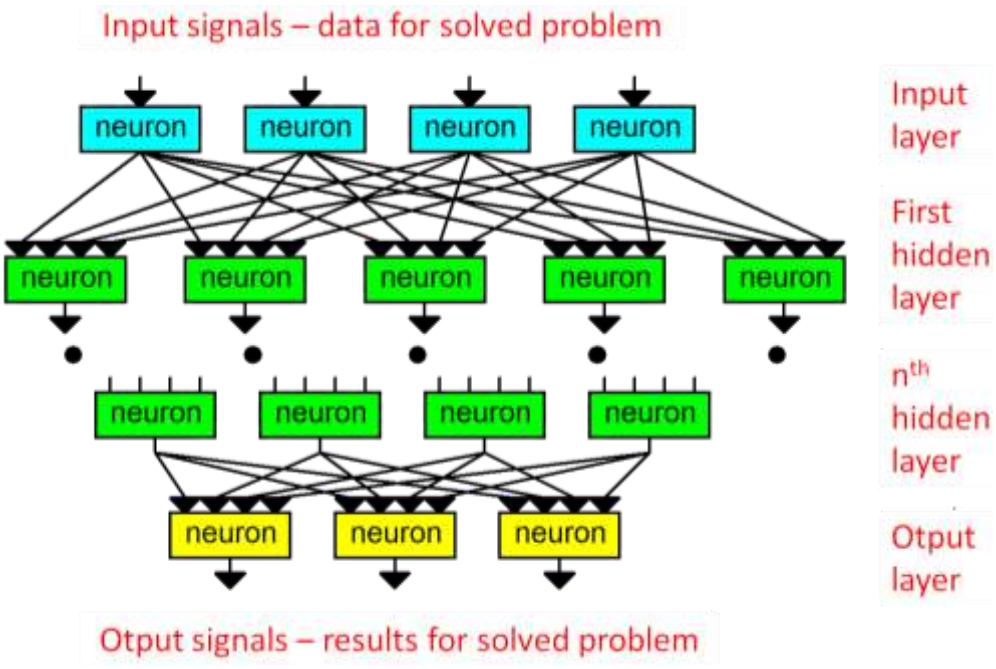


Fig. 7.1. Hidden layers in a multi-layer neural network

What does this problem mean?

The rules of teaching that you have acquired in the previous sub-chapters were based on a simple but very successful method: each neuron of the network individually introduced corrections to its working knowledge (by changing the values of the weight coefficients on all its inputs) **on the basis of the known error value which has been made**. In case of single-layer network the situation was simple and obvious: output signal of each neuron was compared with the correct value given by the teacher which gave the sufficient basis to correct the weights. In case of multi-layer network the process is not so easy. Neurons of the final (output) layer may have their errors estimated in quite

simple and certain way – as previously, by comparison of the signal produced by each neuron with the model signal given by the teacher.

What about the neurons from the previous layers? Here, the errors must be estimated mathematically because they cannot be measured directly – we lack the information what SHOULD the values of the right signals equal, because the teacher does not define the intermediate values, he or she only concentrates on the final effect.

A method, which is commonly used to "guess" the errors of neurons in hidden layers is the method called ***backpropagation*** (backward propagation of errors). This method is so popular that in most ready programs which are used to created networks models and to teaching networks – the method is applied as default method, although there are currently many other teaching methods, for example an accelerated method of this algorithm called ***quick-propagation,*** as well as methods based on more sophisticated mathematical methods such as ***conjugate gradient method*** and the ***Levenberg-Marquardt method***. The mentioned methods (with at least a dozen more which are even more sophisticated) have such advantage that they are very fast. But such advantage occurs only in case when the problem which should be solved by the neural network (by finding out the method of its solution on the basis of teaching process) meets all sophisticated mathematical requirements. However, in most cases when we know the problem which should be solved by the network we do not know if the problem meets such complicated assumptions or not.

What does it mean in practice?

In a nutshell, the following situation occurs:

We have a difficult task to solve, so we take a neural network and we start to teach it with the use of one of those sophisticated and modern methods – for example we use the *Levenberg-Marquardt* algorithm. If we had that kind of problem, how easy the network can be taught with the use of this method – then the network would quickly be well trained. But if not so, and in case such modern algorithm will endlessly lead us astray and the network will not learn anything – it means such theoretical assumptions are not met from the beginning. On the contrary to the previous method, the ***backpropagation***, which I shall present in this chapter, has such nice advantage that it works independently from whatsoever theoretical assumptions.

It means that, contrary to the other clever algorithms which **sometimes** work, the ***backpropagation*** method **always** works. Of course, sometimes it may work irritatingly slow – but it will never let you down. It is worth to get to know the method, because those people who use the neural networks professionally often and willingly come back to this method as to a well-tried out partner.

The ***backpropagation*** method will be presented in action through the analysis of behavior of another program which I will present here. Before it happens, however, we must come back to one detailed issue, which will be very important here and which has been a little neglected so far. I will tackle the problem of the **shape of nonlinear characteristics** used in testing neurons.

7.2. How to change the "threshold" of nonlinear characteristics of a neuron?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

In the previous chapter the analysed neurons were based on ***"all or nothing"*** rule. They could work on the basis of logical categorisation of input signals (***true*** or ***false*** – **1** or **0**) or could be based on bipolar characteristics (***approval*** or ***disapproval***, signals **+1** or **-1**). In both analysed cases the transition between two different marked states had a sudden character: either the output signals (in sum) "exceeded the threshold" and then the signal on the output instantly equalled **+1** or the output signals were so weak ("subliminal stimulation") that there was no reaction (signal **0**) or the reaction was totally negative **(-1**).

What is more, the rule of zero value of threshold was assumed, which meant that the positive sum of input signal equalled **+1** signal and the negative value was **0** (or **-1**) – which limited the possibilities of the analysed networks. By the way, when we discuss the transition of nonlinear neuron, it would be advisable to analyse also the notion of the value of **threshold** because in general cases it needs not to have zero value.

In the non-linear models of neurons analysed today, the **threshold** will be released which will lead to achievement of moving characteristics with the possibility to chose the point of switch freely, with the use of parameter usually called **BIAS** in the programs for modelling networks. With the use of program **Example 07a** you will be able to draw out a family of threshold characteristics of neurons with changing value of BIAS – it will let you achieve a very good overview concerning the role of that factor in the shaping of the behaviour of single neurons and the whole networks. This program shows you what the behaviour of the neuron with freely shaped threshold can look like (fig. 7.2).

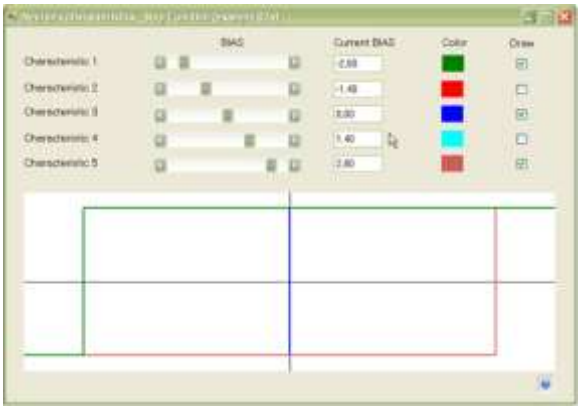


Fig. 7.2. The family of the threshold characteristics of neuron with changeable BIAS value (view from program Example 07a)

7.3. What is the most common shape of the nonlinear characteristics of a neuron?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

The characteristics of the real, biological neuron are even more complex. Between the state of full, maximum stimulation (as physiologists say the "tetanus" stage) and the subliminal state in the form of no activity at all, the so-called "diastole" stage, there are many intermediate stages which may appear in the form of impulses with changeable frequency. In other words the stronger the input stimulus which reaches a given neuron by all its dendrites (entries) the bigger the frequency of the impulses on the output of the neurons becomes. It can then be assumed that in the brain all the information is transmitted with the use of PCM[5] method which was invented by Mother Nature… a few billion years earlier than the engineers specializing in telecommunications!

A full discussion over the issue of coding the signals of neurons in a real brain goes beyond the topic range of this book, but if you are interested in this topic – look through my book entitled *Problems of Biocybernetics.* From the point of view of only this subchapter one conclusion of such analysis comes to mind: it is possible (and purposeful!) to use neural networks built out of neurons on the output of which there are signals **changing in a continuous way** in the range from 0 to 1 or from -1 to 1.

Such neurons with continuous nonlinear transfer function differ positively from **linear** neurons, which we dealt with for a long time in some previous chapters, as well as from **non-linear discontinuous** neurons introduced in the previous chapter which have only two allowed output signals. I am not able to present here all the advantages (particularly when some of them can only be described with the use of some mathematical theorems and I promised not to use even only one mathematical formula in this book), but even just theoretically you may easily guess that nonlinear neurons with continuous characteristics give – considerably – broadest range of possibilities. On the one hand they are **nonlinear** structures, so they can (contrary to linear neurons) form multilayer networks which enable finding (in the course of training) a totally **unlimited dependence between the input and output**. On the other hand, however, the signals in such networks can take **any values** in a fluent way which allows for the use of them in tasks where the calculations result of the neurocomputer should not be limited only to decisions of the "yes-no" type but should define a given value – for example the expected soaring in the stock market in lately fashionable networks which analyse the market and predict its changes.

The second argument is less obvious but as important as the first one and it suggest the abandonment of the simple, "jumping" characteristics. Thus, in order to teach the multi-layer neural network it is **NECESSARY** for the neurons out of which it is built to have **continuous and differential**[6] characteristics. Proving that this notion is really easy would be considerably simple, but I do not think

[5] It is a modern method of impulse coding of signals, used in modern electronics, automatics, robotics and telecommunications.

[6] Both notions – the continuity and the differential are of course mathematical notions and I promised you that there is not going to be any mathematics in this book. But if you do not understand at this point what those notions mean, this fact will not make the understanding of the rest of the text any more difficult, because the mentioned properties will not be further (openly) used.

that this book is the right place to present such proofs. But it is important to remember this fact – the **transfer function of a neuron in a multi-layer teaching network must be sufficiently "smooth"** if teaching should be conducted without interruptions.

We could find many functions suitable for the role of transfer function for modeled neurons but the most popular is the **logistic curve**, called also (from the shape of its graph) a ***"sigmoid"***. A sigmoid has the following advantages:

=> provides a smooth transition between the values 0 and 1;

=> has smooth derivative which can be easily and practically calculated;

=> has one parameter (usually called "BETA") the value of which allows for choosing the shape of the curve freely – from very flat one, close to linear function, to very steep, "relay" transition from 0 to 1.

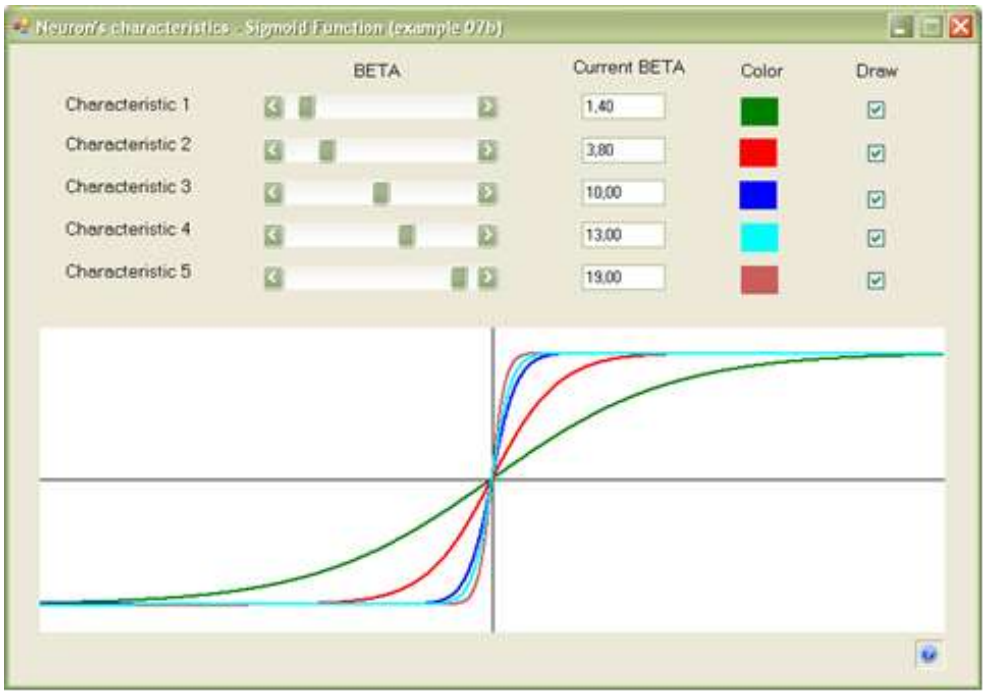


Fig. 7.3. Various forms of logistic curve (view from program Example 07b)

Program **Example 07b** will allow you to get to know the function more deeply. It is easy to notice that by building the nonlinear transition function of a neuron the possibility of moving the "switch" point between the values of 0 and 1 with the use of parameter BIAS is taken into account, but you may additionally choose the steepness of the curve and general degree of the nonlinear behavior of the neuron (fig. 7.3).

Logistic curve, the properties of which you explored in the experiments with the above mentioned program has a lot of applications in natural sciences. I may write something more about it one day but now I would like to say state only one thing: each **development** (for example a development of a new company or technology) goes right according with the logistic curve. Think for a while, and you will see that it is true: at the beginning the growth is small and the development is slow. As you gather experience, the money and other resources the development starts to accelerate and the curve rises up more steeply. In the moment of the biggest success, it the steepest section of the line

going up – a refraction appears and the curve begins to bend towards a horizontal line – at the beginning it is almost unnoticeable but later on radically leading to a stop in growth of the curve, that is to "satiation". The end of each development is stagnation and then – which is not visible on the logistic curve – inevitable fall. But this is a topic for a totally different story…

A variation of a sigmoid for bipolar signals (symmetrically arranged between -1 and +1) is a **hyperbolic tangent**. The name sounds dangerous, but it is really a very nice function. The best option is to see it yourself in the next program **Example 07c**. Was it true that this function is not scary at all?

Once you know what the characteristics of nonlinear neurons look like – it is time to build a network out of them and start to teach it.

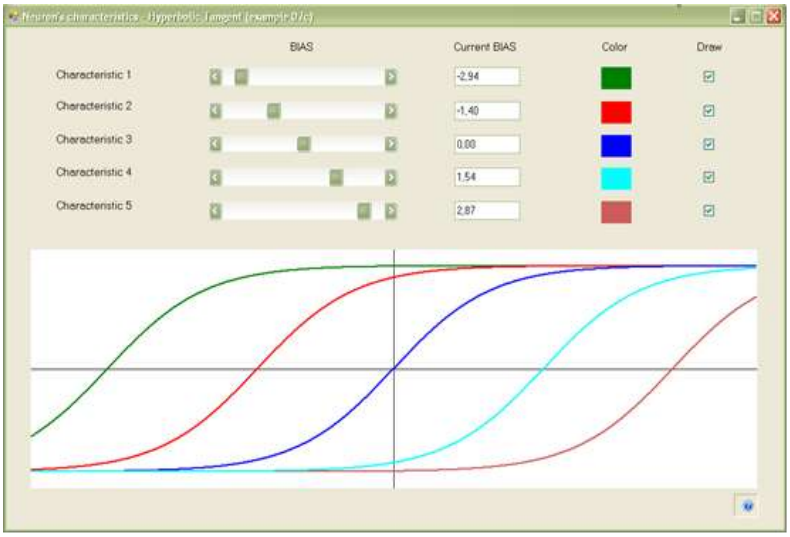


Fig. 7.4. Various forms of hyperbolic tangent function (view from program Example 07c)

7.4. How does the multilayer network constructed from nonlinear elements work?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

Multi-layer networks which are constructed of nonlinear elements are currently the basic tool to use neural networks in practical applications. There are a lot of ready programs which may be successfully used when needed. I will tell you about the ready programs modeling various kinds of networks (both commonly available – *public domain* and a lot of commercial ones) some other time because what I want to present here are two simple programs which will make it visible for you first how a multilayer network works (program **Example 08a),** and then (what you will find in the next subchapter) I will show you the program **Example 08b,** where you will find a method of teaching such a network.

If you analyze those two subchapters carefully and conduct the experiments with the use of offered programs, you will be able to see yourself what this all *backpropagation* method is really about. I really encourage you not to hurry and to read very carefully, think it all over and precisely in person try out all what is going to be presented here, because ***backpropagation* is the key and core of techniques in neural networks**. When you get to know precisely and understand this technique –

most issues connected with networks will be easy and obvious for you. If you do not, however, get involved in this game which I offer to you here then you will not know exactly "*what is going on*". I am sure you can live without it, but is it really worth it?

Before you start the next programs – read a few additional remarks concerning how they work which is not (unfortunately) as obvious as the functioning of the simple programs I showed you before.

Program **Example 08a** shows the functioning of complex network built out of only four neurons really precisely (it will be easier to follow them, see fig. 7.5) which form **three layers –** input, which cannot be taught (at the top of the screen), output, where the signals will be copied on the output of the network where they will be assessed and taught (at the bottom of the screen) and the most important **hidden layer,** presented in the central part of the screen. Neurons and signals in the network will be numbered with two numbers: number of the layer and following number of neuron (or signal) in a layer.

And now a few words about the functioning of the program and instruction of use.

At the beginning the program asks you to give it the weight coefficient for all neurons in the whole network (fig. 7.5).

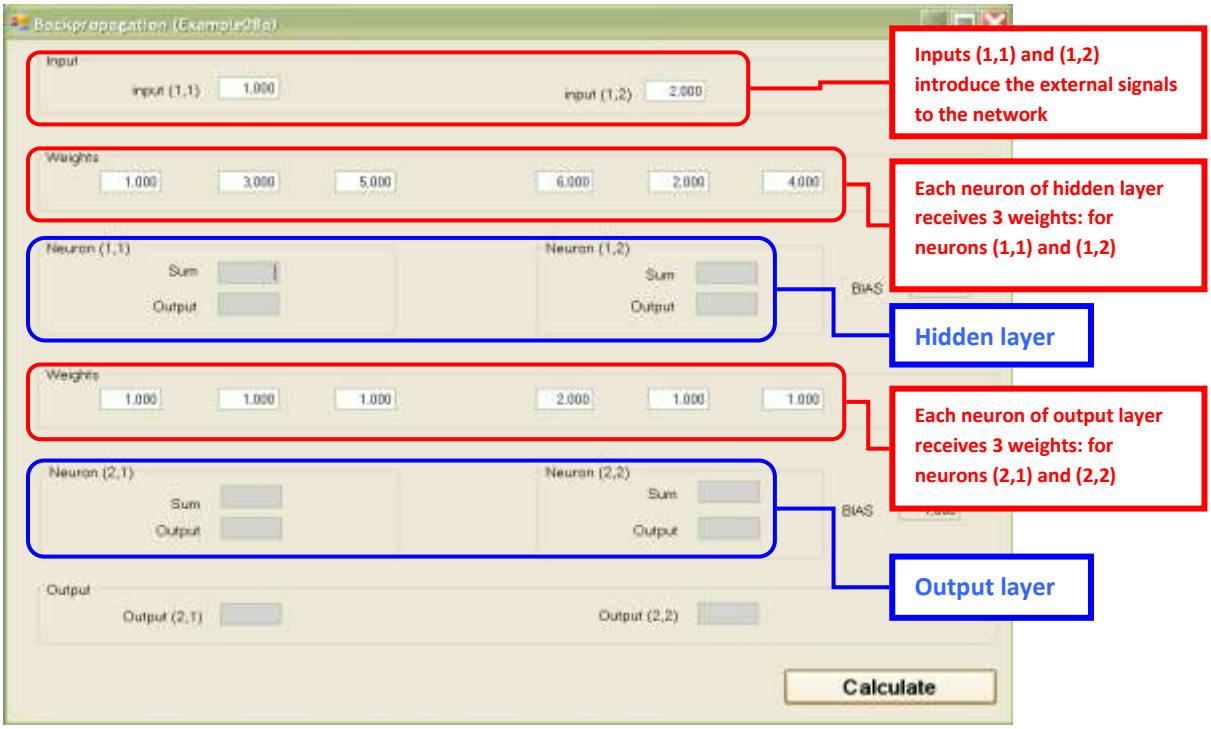


Fig. 7.5. Setting the parameters and input signals for the network in program Example 08a

You should give exactly 12 of them, because – as I previously mentioned – there are 4 neurons being taught here (two in the hidden layer and two in the output layer) and each of them has three inputs (two from the neurons of the previous layer plus an additional input – also possessing the weight coefficient – connected with "the threshold" which presents, in a given network, the generator of the artificial, constant BIAS signal). When you work hard on that, think how nice it is in typical networks, which possess hundreds of neurons and thousands of weight coefficients, that you do not need to write in all the data personally but they are set on their own – automatically – exactly during the

teaching process. In the program described here, I gave you the option of "writing in" all the parameters because I would really like you to have the chance to check if the results which are provided by the network on the output of its neurons are in accordance with those which you think should be right. In order to do it you will be able to shape the input signals to the network freely and observe the signals both on the inputs of neurons and on the outputs. For the lazy people there are of course the default values… It is best to chose simple whole numbers as weight coefficients and input signals because in this way you will easily check if the results which you achieve from the network are in accordance with those which you calculated yourself and you will check if you really know and understand what goes on in the network.

After giving the coefficients you must additionally give the values of both input signals which the network will get on its inputs (fig. 7.5). Now is the moment when the program is ready to work. You must have noticed how the structure of the network is presented (places where particular neurons are placed and where particular signals will be displayed – see fig. 7.6.).

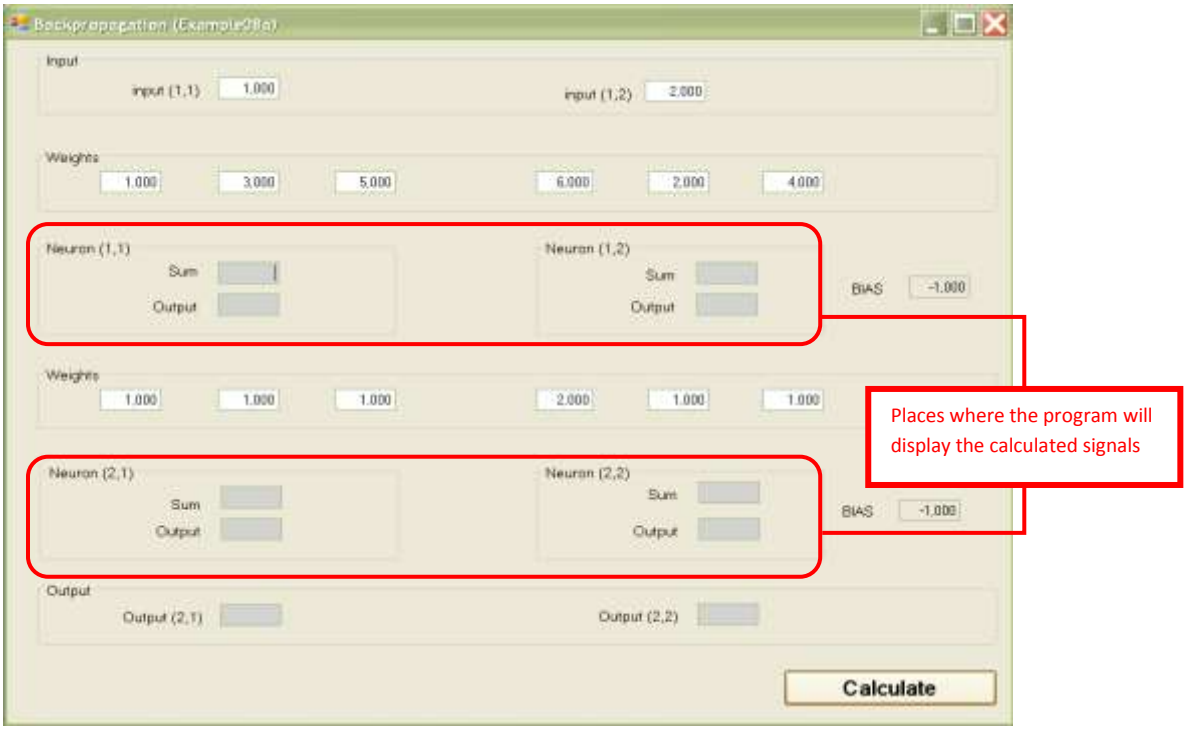


Fig. 7.6. Beginning of the network demonstration in the program Example 08a

If you click the button **Calculate** you start it and you will conduct the simulation process of sending and processing the signals in a modeled network. First – in each following layer there are calculations made and displayed (against a red background) of the **sums of the values of signals multiplied by the right weights** of input signals (together with BIAS component). Next, the calculated values of the **output signals** will be shown ("answers") of particular neurons (also initially against a red background), which were created after pushing the summed input signals through transfer function (with the shape of a sigmoid). The roads of the signals being sent are marked on the picture (fig. 7.7.) in the form of blue arrows.

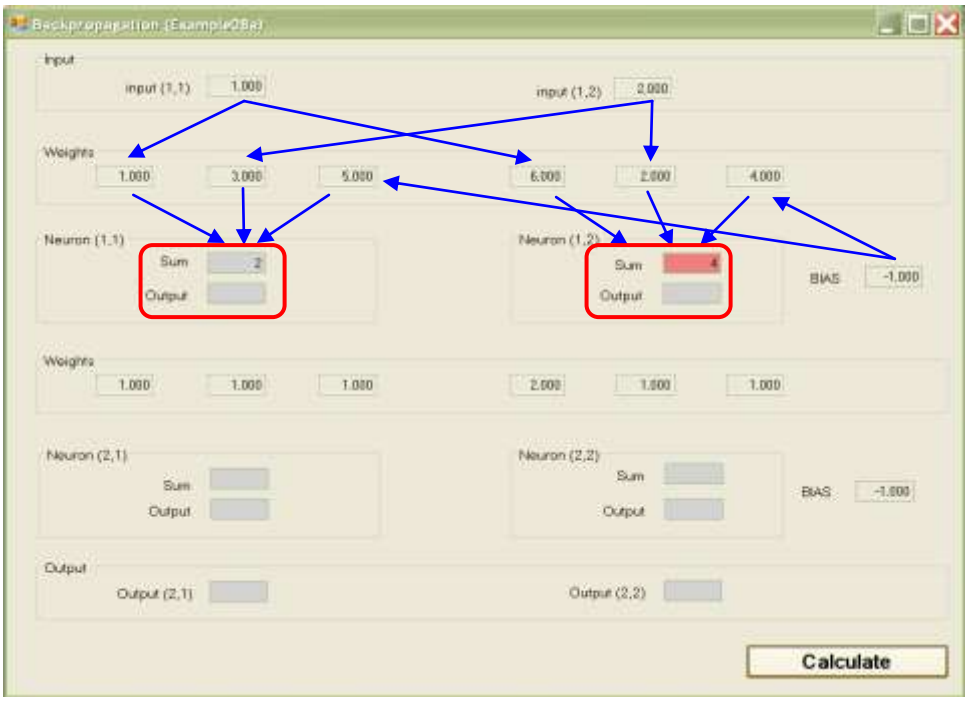


Fig. 7.7. Visualization of the direction of signals transmission in the initial section of the network

Next the answers of neurons of the lower layer become the inputs for the neurons of the higher layer and the whole process repeats itself (fig. 7.8).

If you press the button **Calculate** again, you can observe the movement of the signals through following neurons from the input to the output of the analyzed model of the network. You can change the values on the inputs and observe the outputs – as many times as you want to check if you really understand correctly what really goes on in the network. I advise you to spend some time on detailed analysis and thinking over each number on the screen (you can even calculate it alone on your calculator to be sure the numbers are matching those which you would expect on the basis of your knowledge about neural networks) because only in this way will you know exactly and understand precisely what the network really does.

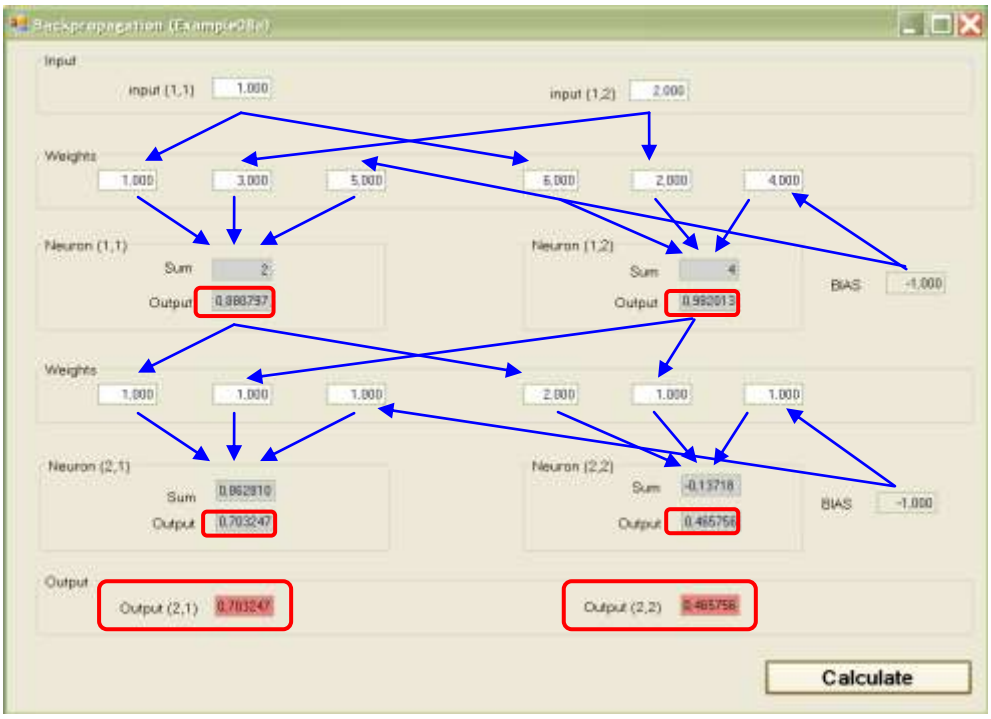


Fig. 7.8. The last stage of the functioning of the program – all signals have been calculated

7.5. How can you teach a multilayer network?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

When the above described multilayer network, in which you personally give the weight values and signals, has no more secrets for you – try and introduce and start the same network but as a network which is **learning**. Program **Example 08b** will do it for you. After launching this program you must give the values of two coefficients deciding about the course of the teaching process. First of them determines the size of the correction which is introduced after finding each of the incoming mistakes and at the same time it marks the total speed of learning. The bigger the value of the coefficient you give, the stronger and faster the changes in the weights of neurons will proceed in the course of detection of mistakes while learning.

This coefficient is called *learning rate* in literature and for this program I called it simple *learning coefficient*. This coefficient must be chosen very carefully because both too big and too small value of this coefficient drastically influences the course of learning. Fortunately, you do not have to wonder what value the coefficient should equal because I have found the right value and the program will suggest you this particular value. But if you want – you can change the given value of the coefficient for any other and then you can be surprised for a long time why has everything "gone wrong" – they say we live in a free country and everyone can do what they want!

Second coefficient defines the degree of "conservatism" of the process of learning. In literature, this coefficient is defined as *momentum*, which means, as can be easily found in a dictionary, a physical quantity of momentum. If you want, you can use a Polish name but don't be surprised later on when

your friends will talk about your suspicious activities connected with physical attractions* towards neural networks. This is why I left the original English name in the program. The bigger the value of momentum coefficient is, the more stable and safe the process of teaching becomes – but too big value may lead to difficulties in finding the best(optimum) values of weight coefficients for all neurons which may be taken into account in a network. Again I have found considerably right value of this coefficient. It is enough now to confirm both offered values by clicking the button **Intro** (see fig. 7.9). You can, however, try out your luck and find a better value. I wish you good luck – you may succeed!

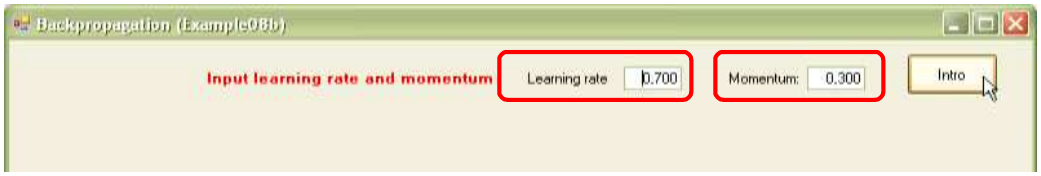


Fig. 7.9. Setting of the parameters marking the efficiency of learning in the program Example 08b

After setting the coefficients, either chosen by you or given to you by me, the program presents you a window, with which you are familiar from previous experiments, and in the window you see particular neurons, values of signals, the weight coefficients and errors. To simplify actions the program has additional element introduced which explains and informs you in the form of subtitles on top of the screen what is going on in a given moment. Such simplification will become useful and necessary because the tasks of this new program will be more complex than those of the previous one – because this program will teach the network.

7.6. What should be observed while teaching a multilayer network?

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

The task of the modeled network is to identify the signals given on the input correctly. Ideal identification should display in the initial layer signal **1** on neuron assigned for a given class (either left or right): second neuron should of course give at the same time the value of **0**. In order to assess the correctness of network functioning it is enough to set some less strict conditions – it is enough when the error on any of the outputs is not bigger[7] to **0,5** because then the classification of the input signal given by the network does not differ from the classification given by the teacher. As a result – the notion of **error** (measured as divergence between the real values of signals on the input of the

* Translator's remark– Polish word "pęd" is close to the word "popęd" in Polish language which signifies physical attraction to something and which in context sounds dubious.

[7] For example, for the input signals which appear is some stage of the learning process which equal **-4,437 and 1,034** the responses of the network are **0,529 and 0,451**, which is qualified as an error because the objet with such co-ordinates should be classified is the second class and in the meantime the input of the second neuron is smaller than the first. On the contrary the second object of the teaching array presented by the program with co-ordinates **-3,720 and 4,937** causes responses of **0,399 and 0,593** and may be considered as indentified correctly (it is also an object of the second class) although the value of mean squared error in both cases is similar (try to observe the similar situation during your experiments!)

network and the values given by the teacher as model values) is not the same notion of **the assessment of the correctness of solving the given tasks by the network,** because this case is connected with the general behavior of the network and not the values of single signals. We can then more often "let it go" for the network, as long as it is able to give you the correct answer. The process of teaching with the use of *backpropagation* method is not likely to be tolerant, as you will see playing with the program. If the result on the output was supposed to be **1** and the neuron set the value of **0,99** – then this is an error and it should be corrected no matter how much the student cries and kicks in fury! (Oh, sorry, I went too far, we didn't plan kicking in fury in our program).

Such perfectionism is, as you can see, worth the trouble because when you make your values of signals perfect according to the suggestions of the teacher the network learns by the way to reach the given aims even more efficiently. A nice analogy with school comes to mind here – although the information received in the process of learning – say geometry – are not directly applied in practice (because in reality we never encounter ideal geometric figures) but the knowledge received for those ideal creations comes in handy for solving practical tasks which the life brings us (for example in order to decide, how much wire net you need to build a fence around a building site with given dimensions). If someone is not working hard when solving seemingly nonsense theoretical tasks, then he or she acquires the basis knowledge in a "so-so" way – he or she then gets into trouble when confronted with real, practical task. The better you learn to solve mathematical tasks, the more efficiently you will manage with problems in practice. It does not concern economy, however, because it is a dominating opinion in Poland that we should strive brutally at maximizing the profit, buying cheap and selling expensive – cheating the client whenever possible. Economy, according to professor Steczkowski from the University of Economy, has nothing to do with school type of mathematics and finds its ideal model only in plundering expeditions of the Vikings. It is then beyond question that learning the idealized physics helps in understanding the very practical techniques, knowledge of biology is helpful when you cultivate the soil and breed farm animals, and the knowledge of geography helps is spatial economics. Neural networks are, as you see, similar in this aspect. If you strive at achievement of **ideal** aims given in the process of teaching by the teacher they come close to the role set for them – the role of **practically** useful tool.

Let's come back to the discussed program. During teaching the network your role will be limited only to observation. It is enough then for you to sit, press any button and watch carefully what is happening on the screen. Everything is generated by the program itself – from the initial values of weights, input signals to the models of correct identifications. Some data is chosen for given networks at random (the initial weights and the input signals) but the models of correct identifications on the basis of which the network will learn are not random! Try and read yourself from the program or from the way it functions what rule works here because I don't want to spoil the fun. Nevertheless, you don't need to know this rule – it is enough if the network will find on its own what is happening – and will classify the signals correctly. You can be sure that the network can do it even if you can't!

And now a few remarks about the functioning of the program and its operation.

At the beginning the program shows only the structure of the network and you are kindly informed about it in subtitles on top of the screen (in red color) – fig. 7.10.

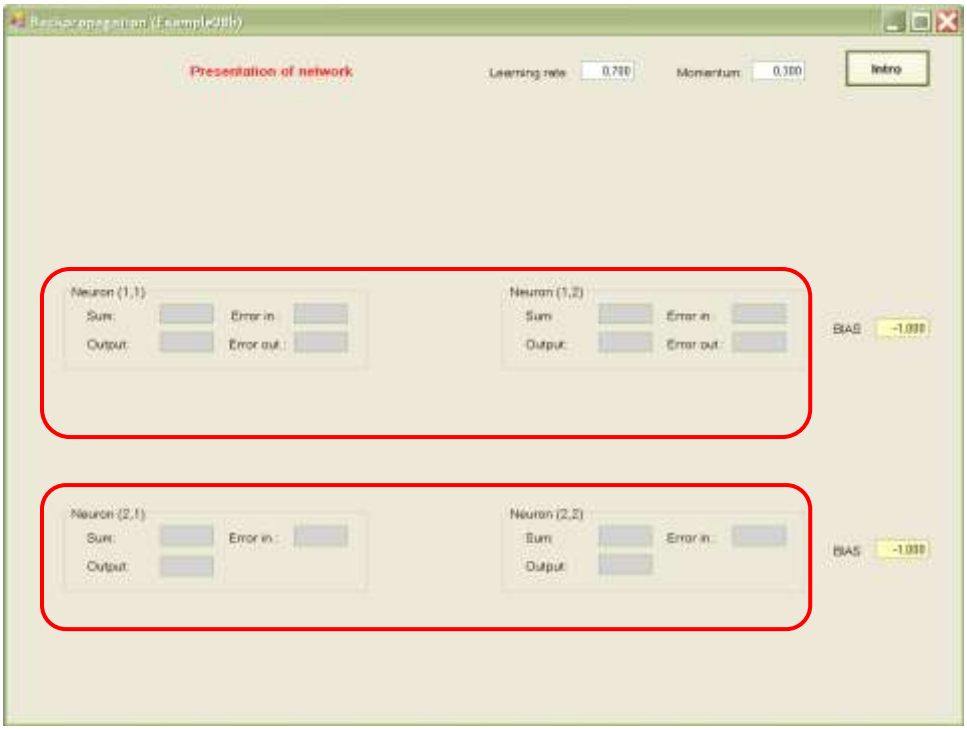


Fig. 7.10. Presentation of the network structure

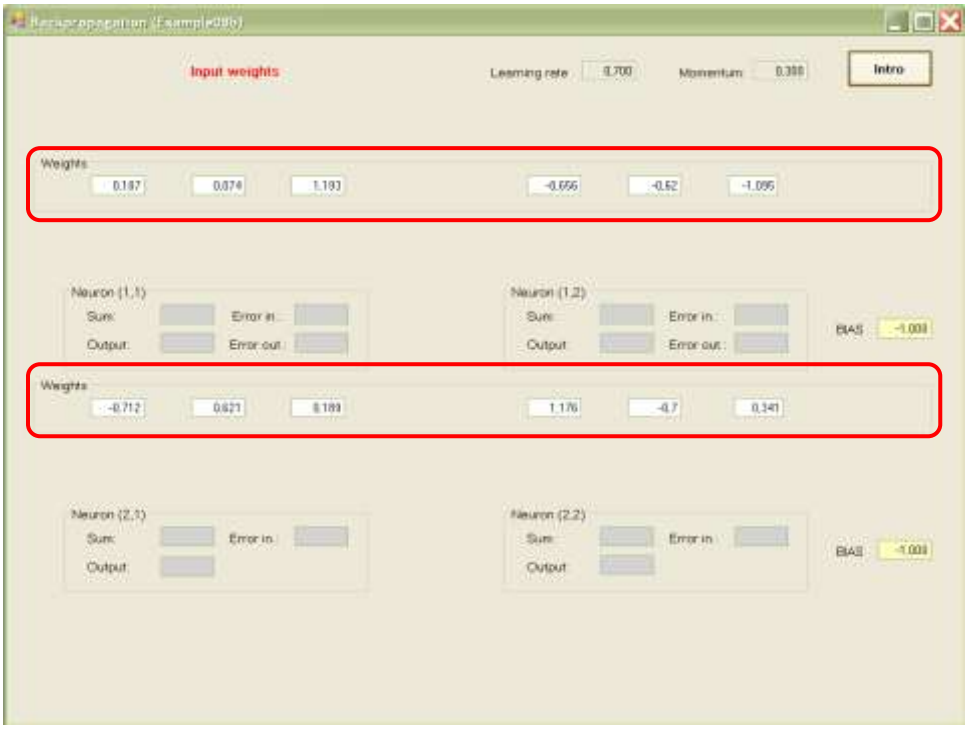


Fig. 7.11. Network and its parameters

Notice that the sources of "pseudo-signals" BIAS were marked this time on the yellow background so that they are not mistaken with the sources (and values) of the real signals taking part in the teaching process. Then, after pressing **Intro** again, the program will show you the **values of weight coefficients** - chosen at the beginning at random for all the input of all the neurons (fig. 7.11).

If you look closely at the network and its parameters – after next clicking of the button **Intro** the **values of input signals** will appear given by the neurons marked with numbers (1,1) and (1,2) – fig. 7.12.

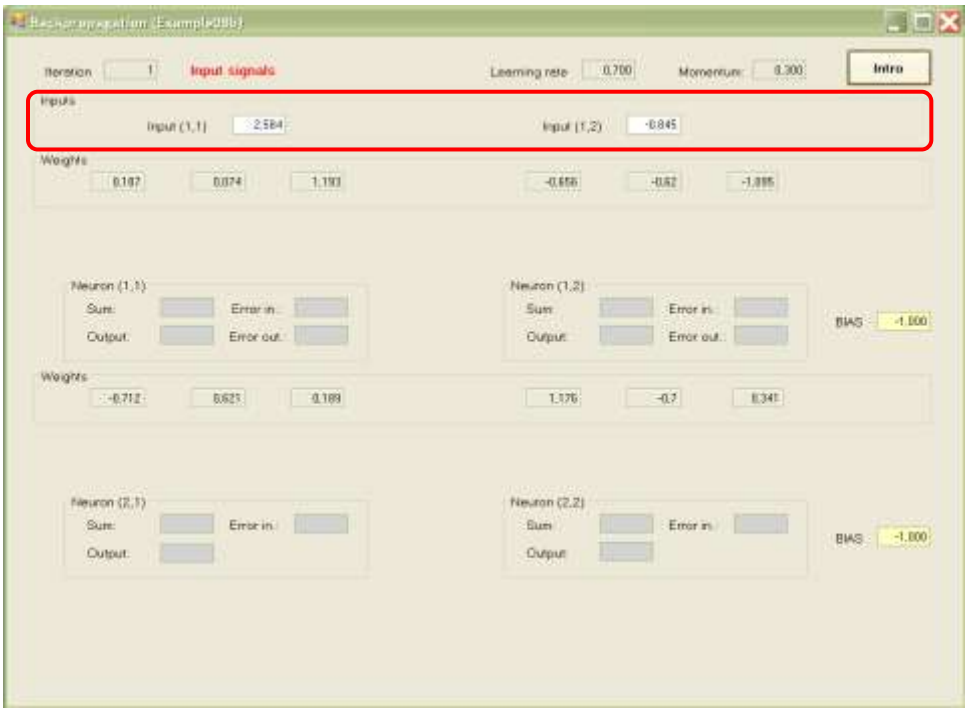


Fig. 7.12. The input signals appear

From this moment up the process of teaching *begins*, but you active role will be limited to clicking the button **Next step** further number of times to start and conduct a simulation of the modelled network (fig. 7.13).

First, the signals calculated during sending signals from input to output will be shown for particular neurons. It is a phase of "forward propagation" – input signals are recalculated by the neurons on the output signals and the process is conducted on all the layers one after another – from the input to the output. This phase could have been observed by you earlier during the experiments with program **Example 08a**. Currently used program allows for observation of the course of the process with the very same precision. First the **sums** of values multiplied by the right weights of input signals (together with the BIAS component) are calculated and then the calculated values of the **output signals** ("answers") of particular neurons are shown (the background is then in red colour) which were formed after sending the sums of input signals through transfer function with a characteristic of sigmoid.

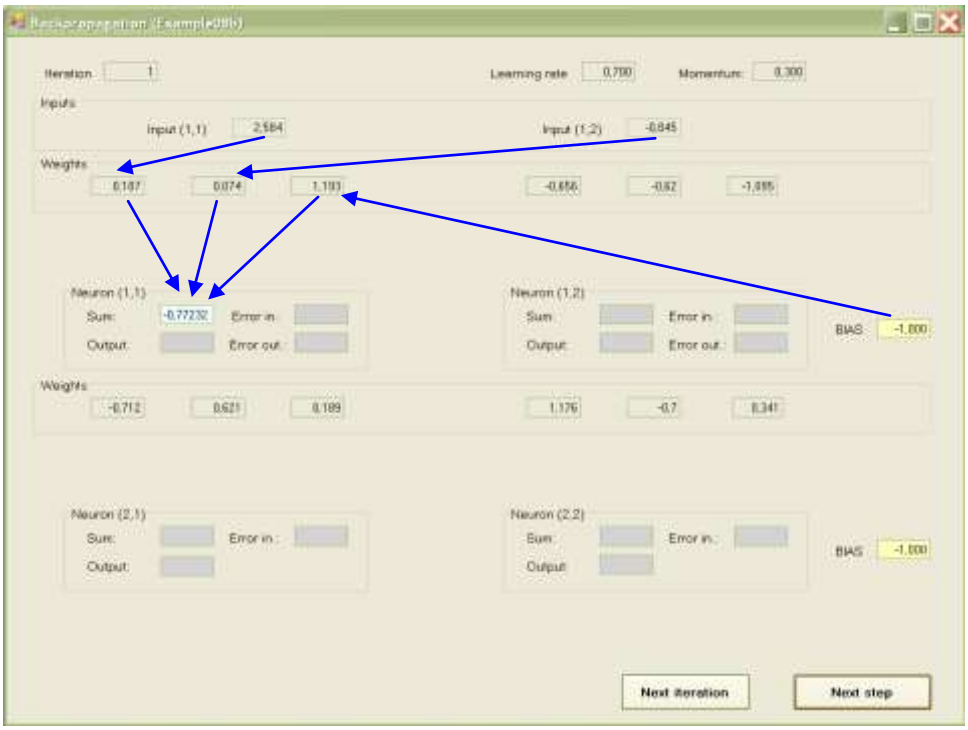


Fig. 7.13. The view of the screen during the initial stage of simulation of functioning of the network

Fig. 7.14. Network after finishing the stage of forward propagation

Of course, the answers of neurons of the lower layer also form the inputs of the upper layer and that is why you can press the button **Next step** and observe the movement of signals through next neurons from input to output of the discussed network model. The target view, after conduction of many steps looks like it is presented in fig. 7.14.

After the output signals are fixed on both outputs of the network (after clicking the **Next step** button again) "the moment of truth" comes – the subtitles at the bottom show you the **given signals** (the models of the correct answers) for both outputs of the network (fig. 7.15) and then the **errors** are marked on the outputs of the network and the **mean squared error** of both outputs is shown. In the end the **final mark** is given for the whole network (fig. 7.16).

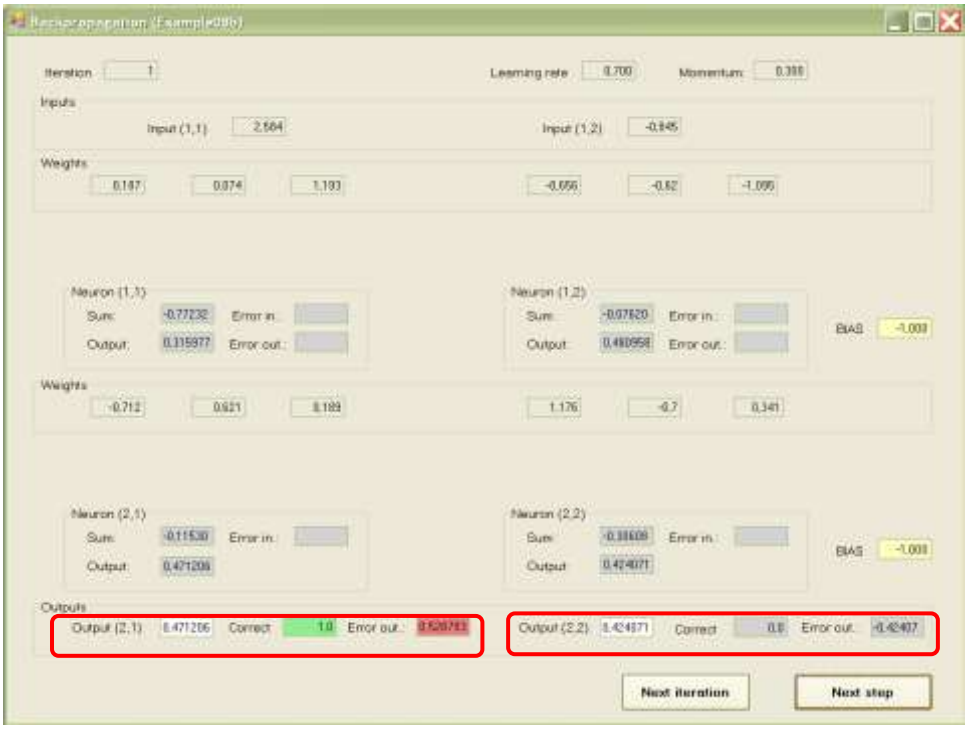


Fig. 7.15. Confrontation of the results of network functioning with the model given by the teacher

At the beginning the mark (given – of course – in red color at the bottom of the screen) will be for sure **"BAD",** but then the network will start to learn and will reach the wanted **"GOOD"** marks. The moment the network (as a whole) makes a mistake – the errors of particular neurons should be calculated.

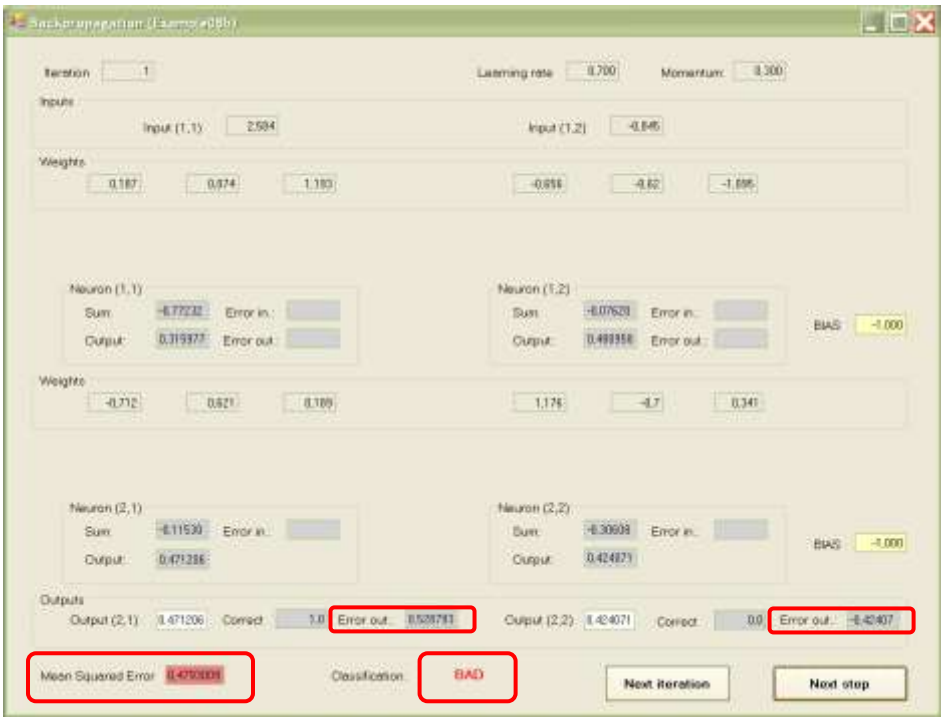


Fig. 7.16. Markings of the errors of output layer neurons and mean error of the whole network

First the error values are marked for the output neurons of the network. Then the errors are recalculated into corresponding values on the inputs of neurons (that is why we need the differentiable transfer function of the neuron). Those **errors transferred to the input** are marked in fig. 7.17.

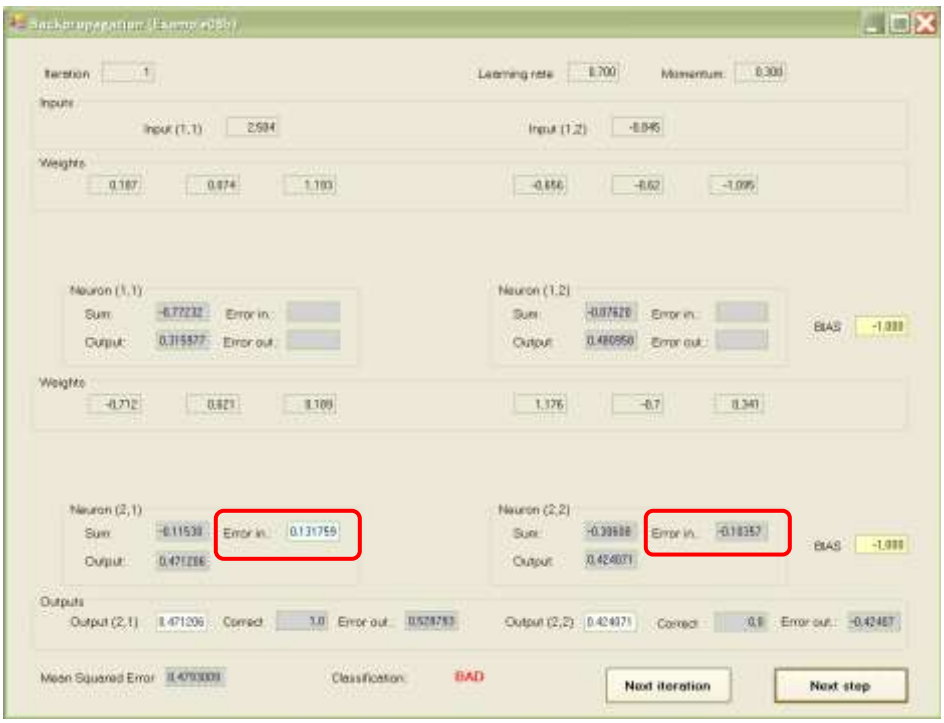


Fig. 7.17. Transfer of errors on the inputs of neurons

Here comes the most important stage – ***propagation of errors to lower (hidden!) layer***. The right values appear by the right neurons – first by their outputs and then by their inputs – fig 7.18.

Fig. 7.18. The result of the backpropagation of errors

When all the neurons have their error values marked - the next step of the program leads them to marking new (corrected) values of weight coefficients in the whole network. **New coefficients** appear below the previously used coefficients and thanks to such solution you are able to look at them closely to see how much and in what direction the process of teaching changed the parameters of the network (fig. 7.19).

It is a very important moment in the experiment with the program. Though you can see a lot of information on the screen, you **can see everything clearly on it** – input signals, output signals, the errors, the old values of the weights and the new ones. Read everything carefully and think it over – the more detailed your understanding how it works is, the bigger chances you have for future successes in applications of networks.

Further experiments with the program can be continued in two ways. If you decide that you want to see the whole course of iteration of the teaching process: the signals going up, the backpropagation of the errors and the changes of weights – you should press **Next step** button. You can also accelerate the whole process by doing next complete iteration that is “jump” way of leading the program to the moment when the new values of all the signals, error and new weights are marked. To do this, use the button **Next iteration**.

Fig. 7.19. Change of weight coefficients

It is the most convenient mode of working on this stage because you will have to conduct a lot of full iterations before the network starts to act in the right way. Due to such solution you can, with the use of the program, easily and comfortably observe the teaching processes, changes of weights and – what is most important in the described algorithm – the process of projecting the error values set in the output layer backwards. You will see and try it out how the exactly marked errors of output neurons and the errors, calculated in the process of backpropagation, of neurons from remaining (hidden) layers enable you finding the needed corrections of values of weight coefficients for all neurons. You will also see that through gradual introduction of corrections the process of network teaching advances. I advise you to analyze it carefully. The example final result of teaching is shown in fig. 7.20.

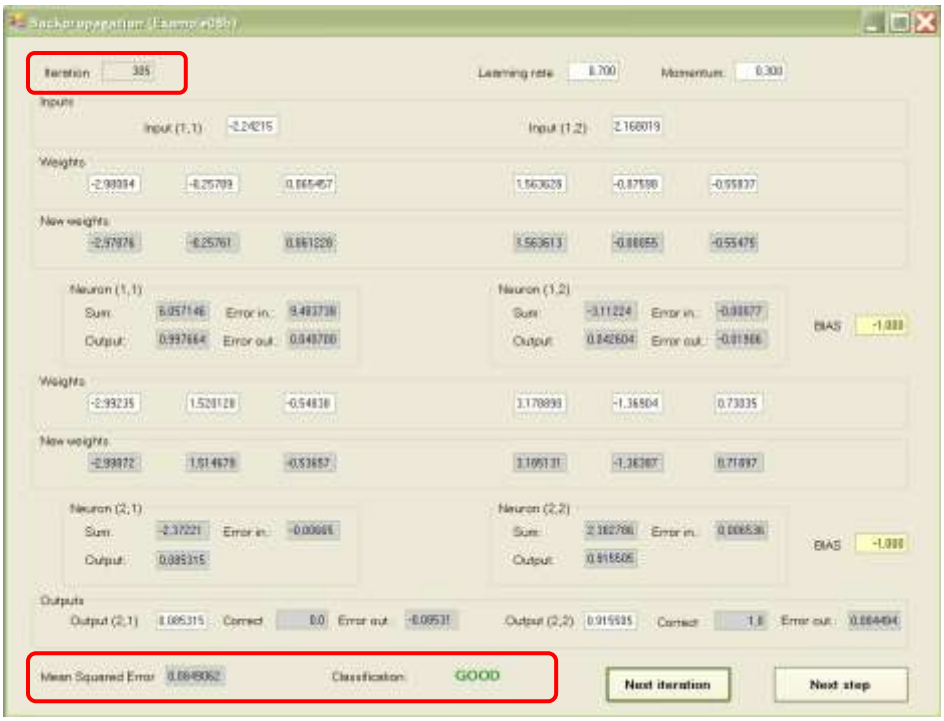


Fig. 7.20. Final effect of the teaching process

Getting to know the program **Example 08b** required some effort, for sure, because is a big and complicated program. But you achieved your own tool (which is easy to modify individually) which helps to acquire deeply and thoroughly the technique of *backpropagation* – one of the most important methods of teaching networks used in modern technology of neural networks. You got also something even more precious – the deep and reliable knowledge about what goes on in the network taught with the use of *backpropagation* method. You will see for yourself how precious such knowledge can be!

7.7. Questions to answer and tasks to be solved individually

(Translation by Piotr Czech, Piotr.Czech@polsl.pl)

1. What is the reason why *backpropagation* is called *backpropagation*? What and why is "sent backwards" in it?

2. What is BIAS and what is it used for? In what way does it the possibility to teach this parameter provided?

3. Why are **two** values displayed in each neuron during the work of the program (**Sum** and **Output)**? How are they connected?

4. What influence on the process of network teaching do the coefficients **Learning rate** and **Momentum** have? Revise the theoretical knowledge you have about it and then try to observe the influence on those parameters during the experiments conducted with the program.

5. What does the process of sending errors from the output of neuron (where it is marked) on its input (where it is applied for weight modification) mean?

6. Which of the neurons of the hidden layer will be most heavily loaded with the error made by a given neuron of the output layer? What is that dependent on?

7. Is such situation possible when the neurons of the output layer show errors and some neuron of the hidden layer has the zero error assigned (by *backpropagation* algorithm) and does not change its weight coefficients?

8. How is the mean mark of the network functioning given? Can we assess the functioning of the whole network in some other way (and not single elements analyzed individually)?

9. **Task for the advanced students:** Add a module to the program, which will show the changes of error values of the whole network and particular neurons in the form of graphs. Is it always so that if the functioning of the whole network is improved the errors made by single neurons of the network are minimized?

10. **Task for the advanced students:** Try to expand the programs described in this chapter in the direction of using more input and output signals and bigger number of neurons, formulating (if you want to treat this task really ambitiously) more than one hidden layer.

11. **Task for the advanced students:** With the use of network which uses bigger number of inputs, try to observe the phenomenon of throwing out the unnecessary information in the course of learning. To do so, give unnecessary information (such which have no connection with the required response of the network for example values from the generator of random numbers) to one of additional inputs. The other inputs should get all data required to solve the required task! You should observe that after a short period of learning the weights of all connections leading to such “parasitic” input to hidden neurons will take values close to zero and it means the idle input will practically be “amputated”.

12. **Task for the advanced students:** The programs described in the paper presented the functioning of network displaying in all places **the values** of the appropriate parameters and signals. Such method of presentation made it possible to check (for example with the use of calculator) what the network does and how it works, but the program was not very clear and readable in quality observation of the courses of processes. Design and produce such version of the program which presents all values in graphic form which is easier and nicer to interpret.

8. Forms of neural networks learning

8.1. How to use a multi-layer neural net for recognition?

(Translation by Piotr Ciskowski; piotr.ciskowski@pwr.wroc.pl)

Multi-layer neural nets, which you have studied thoroughly in the previous chapter, may be used for many tasks. Nevertheless, if we want to analyze their properties and the way they work, the most appropriate area is image recognition. Image recognition is a problem, in which a neural net (or other kind of a learning machine) decides on the membership of images to certain classes. The analyzed objects may be of various kinds – from digital camera images to scanned or grabbed analog images. We present a comparison of a digital and analog image in fig. 8.1, in order to illustrate what we are talking about, but also to displease all those readers too much confident that "digital" always means "better".

analog image

digital image

Fig. 8.1. A comparison of an analog and digital image

This book is about neural nets however, not about images, so we are not going to go deep into the theory of image recognition, especially considered as picture recognition. Still it is worth noticing that the problem of recognizing images, namely pictures, actually started that field of research and gave name for it. The first (historical!) neural net built by Frank Rosenblatt (presented in fig. 8.2) was used to recognize images and that is why it was called a "Perceptron". Please regard this picture (even if quite ancient and of poor quality) with the proper respect, as a relic of one of the first achievements in the discipline we are studying in this book.

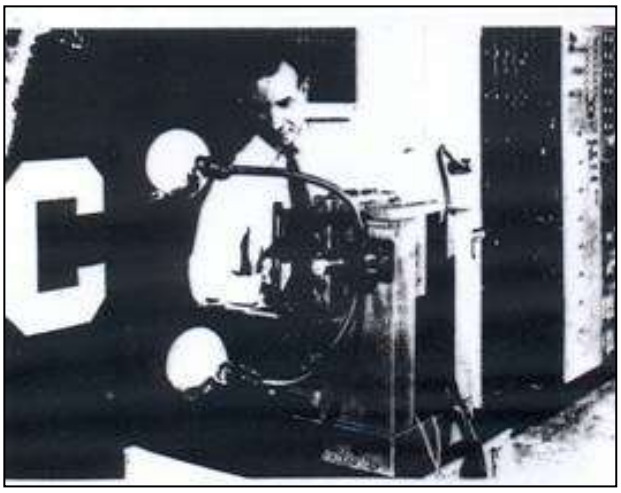


Fig. 8.2. Rosenblatt's „perceptron" – the first neural network recognizing images

The meaning of the word "image" has been generalized so much that now neural nets are used for recognizing samples of sound signals (e.g. spoken commands), seismic or other geophysical signals (when searching for geological ledges), symptoms of patients to be diagnosed, scores of companies applying for loans – and much more. We consider all these tasks as image recognition, even if the above mentioned "images" are in fact, respectively: acoustic, geophysical, diagnostic, economical, or other.

A neural net used for image recognition has usually got several inputs, supplied with signals representing features of the objects being recognized. These may be for example some coefficients describing the shape of a machine's part, or the liver's tissue texture. We often use many inputs as we want to show the neural net all the features of the analyzed object, so that the net is able to learn properly how to recognize it. However, the number of image features is much less than the number of image's elements (pixels). If you supply the net with the raw digital image, then the number of its inputs will go into hundreds of thousands, or even a few million! That is why we practically never use neural nets for analyzing raw images. The nets usually "see" the **features** of the analyzed image, extracted by other independent programs outside the net. This image preparation is sometimes called "preprocessing".

A neural net used for recognition has usually got several outputs as well. Generally speaking, each output is assigned to a specific class. For example an OCR (optical character recognition) system may use over 60 outputs, each assigned to a certain character – e.g. the first output neuron indicates letter A, the second neuron – letter B and so on. We discussed that in chapter 2, if you would like to recall the possible output signals of a net used for recognition, go back to figures 2.30, 2.31 and 2.33.

There is usually at least one hidden layer of neurons between the input and the output of the net – we will now study the hidden layer in a more detail. Generally speaking, there may be many neurons in the hidden layer or just a few of them. After analyzing briefly the processes going on in neural nets, we usually think that more neurons in the hidden layer give a more "clever" net. However, you will soon learn that it is not always worth having a net with large "built-in intelligence", as it sometimes turns out to be surprisingly disobeying!

8.2. How I implemented a simple neural net for recognition?

(Translation by Piotr Ciskowski; piotr.ciskowski@pwr.wroc.pl)

We are now going to build a sample net with just two inputs. Fairly speaking, it is rather not possible to recognize anything based on just two features, and so neural nets are most commonly used to analyze multidimensional data, which need a net with more than two inputs. However, using two inputs in our example has one significant advantage: **Each recognized object may be represented as a point on a plane.** The first and the second coordinate of that point indicate the value of the first and of the second feature, respectively.

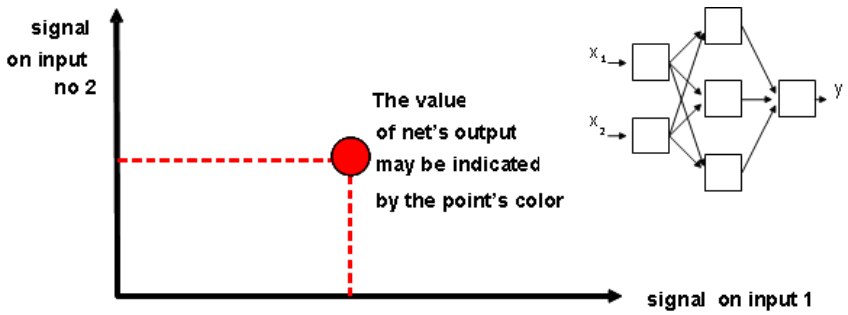


Fig. 8.3.The way we present signals to the neural net

I hope you remember a similar situation in chapter 6.4 where I suggested you imagining that a two-dimensional input space is a brain of an animal having two receptors – e.g. primitive eyes and ears. The stronger the signal captured by the eyes – the point on the plane is more to the right. The stronger the sound – the point is higher on the plane. Please recall now that analogy and fig. 6.8 as it will be very useful also here.

Due to this commitment every image that we show to the net (each „environment" in which an "animal" appears) may be shown as a point on an axes, or as a pixel on a screen, with coordinates corresponding to the features of the considered environment. We will use a screen of a fixed size, so first we should limit the values of our features. In the example considered here both features of the analyzed image will take values from -5 to +5. You will have to remember that scale when formulating tasks for the net.

Each net has an output on which we observe its behavior. In our example there will always be **one** output. We have already interpreted its meaning in chapter 6.4, fig. 6.8. Let us recall: To each presented situation, the animal may react with positive or negative attitude, indicated by the intense red or blue color of our point, respectively. You will place our "animal" consequently in various environments, by supplying the net with various sets of input signals. Those signals correspond to different positions of the "animal" in the feature space (that is different features sensed by the animal). You will be able to prepare a map of places , where your "animal" feels good, and where it doesn't. Such a map should be built of separate points, as it will be created by the program we are going to run soon (see fig. 8.4), yet for our comfort and to improve the visual appearance we will draw these maps a little "smoothed" (see fig. 8.5).

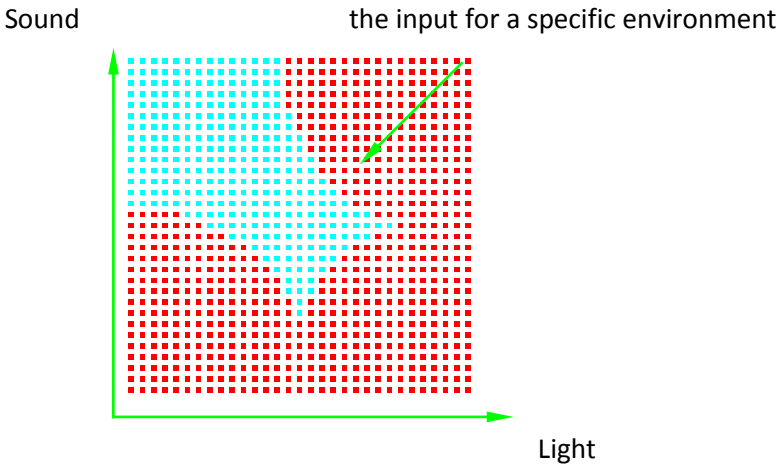


Fig. 8.4. An example of a „map” showing the “animal”
which conditions (regions of input space) it should accept
and which ones it should reject

Fig. 8.5. The same „map” as in Fig. 8.4., but smoothed a little

As the net grows, we will more often observe intermediate states: partial disgust (light blue), a neutral attitude (light green) and partial applause (yellow turning into light red) – fig. 8.6. Just like on a map in geographical atlas – from the deep abyss of blue to the dark red of high enthusiasm!

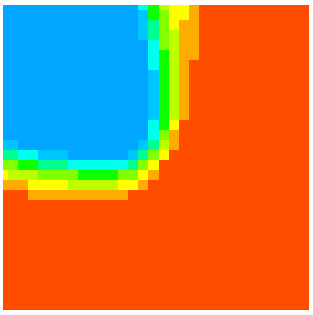


Fig. 8.6. A sample “map” of the trained “animal” at some stage of training

Due to the fact that the net has only one output, it will be easy to show how it **should** work. By switching on a pixel corresponding to a certain input vector, you will see if that point is accepted

(red) or rejected (blue) by the net. Of course the data describing these points will have to be prepared by the program, as I will show further.

You will see maps like these during net's training. They will change many times, as the net will dislike something it approved some time ago or it will convince itself to something it used to reject. It will be very interesting to watch how the initial confusion turns into working hypothesis and then crystallizes into absolute certainty. It is not easy to illustrate these phenomena so suggestively for a multi-output net, but as I mentioned earlier in real systems you will usually see more than one output.

Using only one output in our example has another advantage: you don't have to worry which net is better – the one which classifies a signal into several classes at a time with the same confidence (while the proper class is one of all those indicated) – or the net with all responses weak, among which the strongest one is the correct class of the analyzed object. Multi-output nets may cause many other problems; therefore it is hard to judge the current performance of the net, its reaction to unseen situations (thus its ability to generalize its knowledge). In a one-output net the situation is always clear.

8.3 How to choose the structure of the net for our experiments?

(Translation by Piotr Ciskowski; piotr.ciskowski@pwr.wroc.pl)

Program Example 09 allows you to experiment with the net. At the beginning the program shows a panel, in which you may define the structure of the net to be tested. The program will suggest some default parameters, which of course you may change, and that is what we are going to do at the beginning.

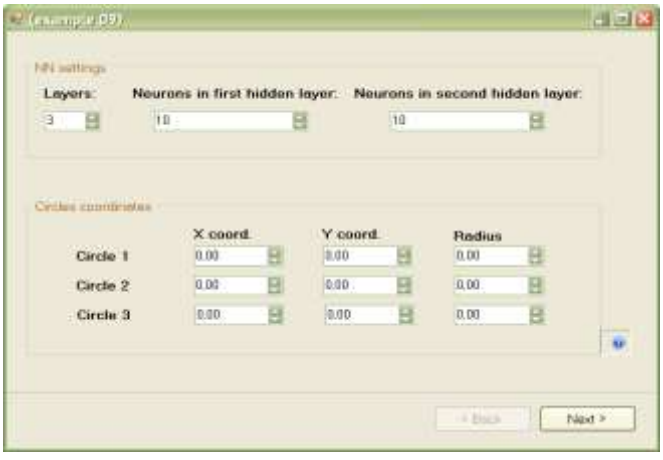


Fig. 8.7. Defining the net's structure in **Example 09**

We have agreed that the net you are going to teach and analyze will have the structure:

2 – xxx – 1,

where the successive numbers stand for the numbers of neurons in the following layers: two inputs, one output, and some neurons (xxx is a number to be modified) in the middle. To define a specific

net, you will only have to specify the number and the order of hidden neurons (that is how many hidden layers to create).

You will be able to define the number of neurons in each layer of the net, depending on how many layers you choose in the field **Layers**. When specifying the number of layers, remember that only these layers count which have neurons that are able to learn – that is why the output layer counts, although it contains only one neuron, while the two input neurons, which only accept input signals, do not count. Thus if you choose to have a one-layer net, you will get such a net:

2 - 1

and you will not be able to specify the number of hidden neurons as this number is already fixed by net's architecture. If you choose a two-layer net, you will get the following structure:

2 – x – 1

and the field **Neurons in first hidden layer** will appear, where you will be able to specify the number of hidden neurons – see Fig. 8.8.

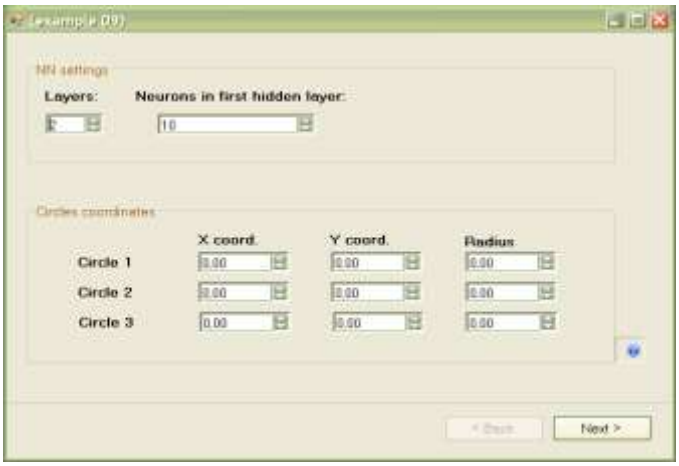


Fig. 8.8. Defining the net's structure in **Example 09** – in case of a two-layer net

Finally, if you choose a three-layer net (or just leave the default settings unchanged) you will get such an architecture:

2 – x – y – 1

and you will have to set the number of neurons in both hidden layers (fields: **Neurons in first hidden layer** and **Neurons in second hidden layer**).

Our program cannot simulate more than a three-layer net. You may change this significant limitation by yourself if you want to "play" a little with the source code. We chose to limit the number of layers as for big nets the program works slowly. You will see that a few hundred training steps (which is not much when observing the training process) will obviously take some time – even if you have the latest and fastest computer.

8.4. How to prepare recognition tasks for the nets?

(Translation by Piotr Ciskowski; piotr.ciskowski@pwr.wroc.pl)

After choosing the size of the net, and – if necessary – number of neurons in all its layers, we may proceed to the most interesting part – preparing the task to be solved by the net.

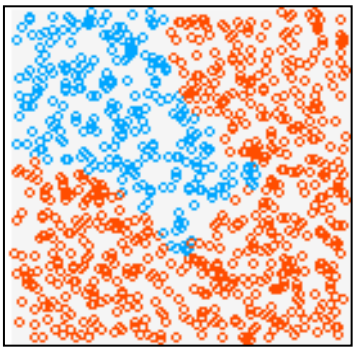


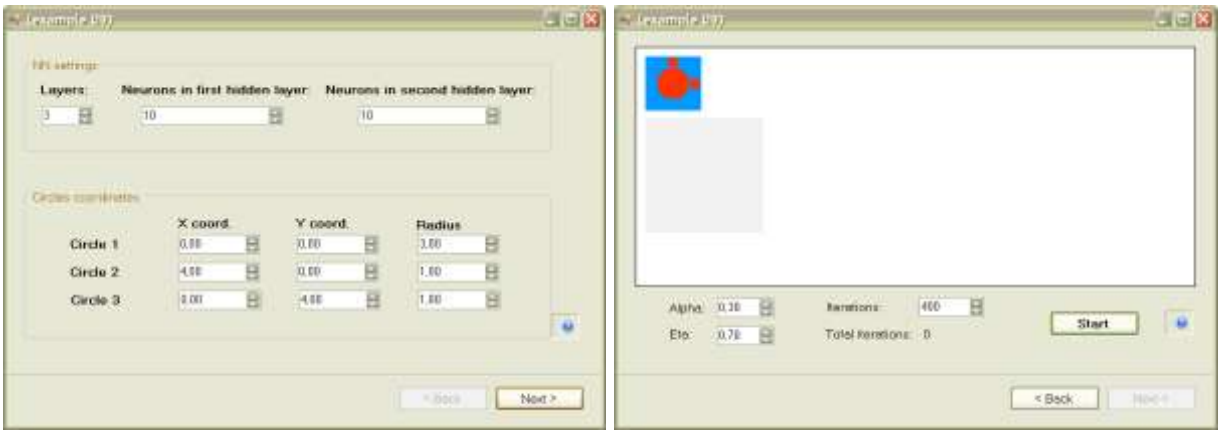
Fig. 8.9. A sample training set generated by **Example 09**

As you know, the idea of recognition problem is that for some combinations of input signals the net should react positively ("recognize" certain situations as nice images), while negatively for other combinations ("reject" those images that it doesn't like). The information about what our net should like and what it should dislike – is contained in the training set. It consists of a certain number of points on a plane. Each of them has two coordinates – so that we know where to put a point on the screen, and the expected answer of the net: positive or negative – so that we know what color this point should be: red or blue.

It is You that decide – where to put blue and red points, that is the desired areas of acceptance and rejection. I could suggest you prepare these data manually as sets of three numbers – two inputs and one output, but it would be tough for you to create a proper task this way, and the final image - the desired net's "preference map" - would be hard to interpret.

That is why I decided that the program **Example 09** will generate examples automatically, while your task is just to show it the positive and negative areas. Again – I decided for You. I could have left the decision on these areas up to You – so You would specify the shapes of these areas only up to yourself – but that would require including a graphical editor in our program, which would get complicated and hard to use. That is why we've got to find the halfway solution – the areas of positive decision, that is the fragments of input space, to which the net should respond "yes" (red points on the plot) will belong to three circles. This is quite a big constraint, but also a fast and comfortable way of preparing tasks for the net. You will be able to prepare really interesting problems for the net, by combining the areas of three circles, of any size and position.

You may set the coordinates of these three circles in the same window (the first one that appears after the program **Example 09** starts) in which you defined the net's architecture (fig. 8.10a). First two coordinates x and y define the centre of the circle, while the third coordinate defines it radius. You may set them independently for each of the three circles (Circle 1, 2, 3).



a) **b)**

Fig. 8.10. a) An example of defining the net's structure and the regions for recognition, b) A "map" of regions, for which the net's should react with a positive and negative answer (after training)

You are free to choose these parameters, but - as I have already mentioned – only that part of the plane is used in the recognition process, for which both coordinates are in the range (-5,5). Therefore quite a reasonable area for placing the points to be positively recognized is a circle with parameters

0,0,3

while you will not be able to such a circle in the screen

10,10,3

so it doesn't seem to be useful in our experiments.

After setting the parameters of the task to be solved by the net, you may see the result of your choice in the next panel, in which your net is simulated. To proceed to this panel press the Next button at the bottom of the window, see fig. 8.10a. The results of your decisions are illustrated by the "map" of positive and negative response areas of your net.

This way you may easily check, whether your expectations on the areas to be learned by the net meet the reality or not. If you are not able to specify any task convenient for you – you may always go back to the previous panel by pressing the Back button, where you will be able to modify the centre coordinates and sizes of the circles.

Training will proceed in the following way. A random point will be chosen, for which the class will be decided according to your map. Then a set of two input coordinates and the desired class will be supplied to the net as an example from the training set. This procedure will be repeated many times.

Let us notice one detail, easy to be missed out. If you define the decision areas in such a way, that most of the decision plane are positive (red points), then during training the net will get mostly these examples while quite rarely it will see the points for which the desired answer is negative, and so it will slowly learn the negative reactions. The same will happen in the opposite direction, if the net is trained with oversized amount of negative examples (the "big blue"). Therefore we should have approximately the same amounts of red and blue training points in our training set. Then the training process will be the most effective. If you don't listen to my advice, the net will finally learn the problem, but you are going to lose much of your valuable time.

After specifying the task as a map that is satisfactory to your ambitions and well balanced (in the sense described above), you may start the training of your net. To do this just press the Start button, as in fig. 8.10b.

At the beginning you will see the decisions of the net with random weights of all its neurons. Usually this initial distribution of net's decisions has nothing common with the task you defined. How would it be possible that before training the net knows what you want from her? However, it is useful to compare this initial map with the map of the task. The whole training process will strongly depend on how similar these two maps are at the beginning. If they look alike, the net will learn fast and effectively. If there are significant differences between these maps (and that usually is), the net will learn slowly and especially at the beginning no major improvements will be seen, as the net will have to prevail its inborn intelligence (which is really a hard thing to do), and only after that it is done it will be able to start improving its knowledge (fig. 8.11)

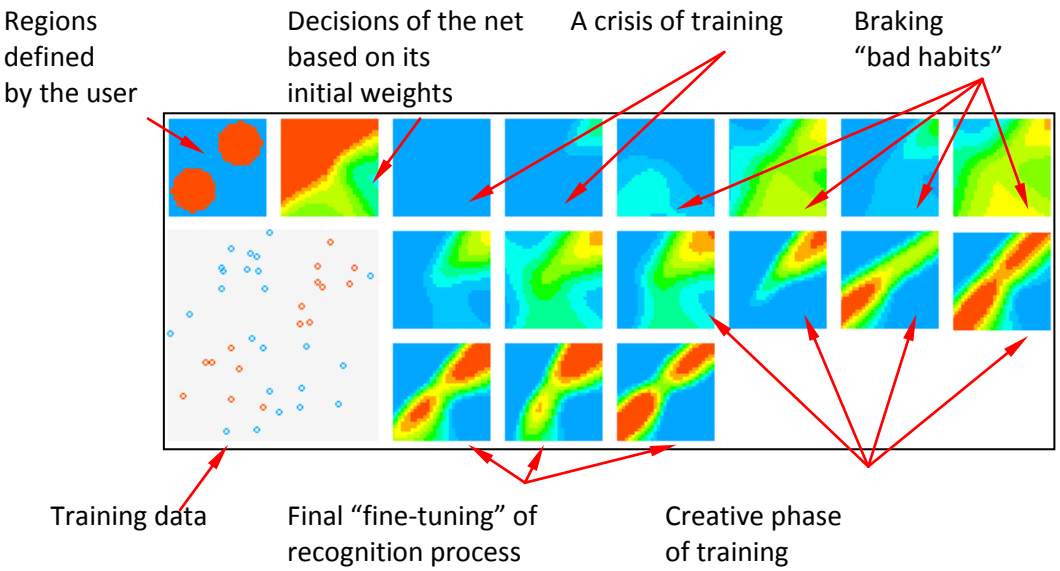


Fig. 8.11. The way **Example 09** presents the subsequent phases of training

Let us say a few words about the structure of the figure displayed by the program. Small rectangles filled with color spots that appear during the simulation are the illustration of neural net's consciousness in each training state. As you already know, each point inside the rectangle represents two features (corresponding to both coordinates of the point) being the input signals for the net, while the point's color stands for the actual net's answer for this input. Let us consider the first two

rectangles, in the upper left corner of the figure. The first one shows what You want to teach your net. Acting the way we described in the preceding section, You assume what conditions are to be accepted by your „animal“, and which are to be omitted – these regions are illustrated by the map of desired reactions. The second rectangle marked out (the second one in the upper row) shows the natural („built-in“) net's predispositions. In each experiment the net may have different initial parameters (set randomly), thus its initial behavior is totally unpredictable.

The following diagrams correspond to the subsequent stages of the training process. They are drawn as rectangles, shown in fig. 8.11, and may be shown in different time steps, as it is You that decide (using the field **Iterations**) on the number of steps between showing the results of training. As You may have noticed, there is an additional read-only field under **Iterations**, called **Total Iterations**, which shows the total number of training steps performed in all training stages until now.

Before performing the given number of steps (specified by You, or left unchanged from the preceding stage), You may also change the values of training parameters – *training coefficient* ***Alpha*** and *momentum coefficient* ***Eta*** – in the corresponding fields shown all the time on the screen. You may set them to any values, but I advise You to leave them at their defaults and not change during training. Then, after You learn the ups and downs of the training process – You may try to modify these parameters and see the results.

8.5. What forms of learning may we observe in the net?

(Translation by Piotr Ciskowski; piotr.ciskowski@pwr.wroc.pl)

The program **Example 09** allows You to investigate many aspects of learning, that may occur in neural nets. These experiments are so interesting and inspiring that I have also spent many hours setting different tasks for the net and analyzing many combinations of training parameters. Let me now share my observations and remarks with You. Then You will be able to tell me how your nets behave.

Learning processes in living organisms, particularly neural mechanisms determining and controlling them, have always been a key interest of biological scientists. Through the years they have gathered and organized observations about learning of humans and animals, in order to discover the basic rules of this process. They have already been used in teaching people and training animals. However, this knowledge is behavioral – based on observing behaviors, without identifying the mechanisms controlling them. Thousands of neurophysiological and biochemical experiments conducted later on in order to discover the idea of learning – gave far less satisfactory results. Luckily, new methods have emerged that enable us to verify the concepts on how the neural system works. Many of these methods simulate the process of learning using neural nets. They use programs like our **Example 09**. It simulates the training process and comes out with figures that bring many interesting ideas to our mind. We shall consider one such case, while further examples You will find out and analyze yourself.

As You remember, training a neural net means submitting examples of input signals (chosen randomly from the training dataset) on its inputs, and forcing the net to produce an output accordingly to the map of preferred behaviors. This scenario may be converted to the previously considered example of an “animal“ with two senses. Such an animal would be repeatedly put into various “conditions” which provide our “animal's” senses different random (but known) signals. The

"animal", or the neural net, reacts accordingly to its current knowledge – it accepts some conditions while rejects the other. The "teacher" (that is the computer which trains the net) having the map of preferred reactions, supplies the net with the desired output signal as if it was telling the net: *You ought to like this and dislike that!*

You may watch the training process dynamically on Your screen, as it is fully illustrated by the big square in the lower left corner of the figure presented by the program. In this square the subsequent points are drawn corresponding to the input signals presented to the net. As You can see, they are randomly chosen from the whole area of the square. Each point is marked with a color – red or blue. This is the teacher's suggestion: this one is good, that one is bad. The net uses these instructions and corrects its errors by adjusting its parameters (synaptic weights). The adjustment is performed according to the backpropagation algorithm presented earlier.

The training process is paused once in a while and the net is examined – it has to give its answers for all the input points. The results are presented by the program using the "maps" of color points, arranged as the consecutive squares from left to right (as in cartoons) and from upper to lower row of the image produced by the **Example 09** program. After each such pause You decide on the number of steps until the next exam.

I will begin my presentation of the ways neural nets learn with an example of such a one-layer net:

2-1.

The task for this net is very simple – to divide the area of all possible input data into two regions: approved and rejected, using an almost straight line. To define this problem the following coordinates of three circles were defined in the program:

100,100,140

0,0,0

0,0,0

We defined a map of "preferred" net's behaviors, which is simple and easy to interpret (see fig. 8.12) – our "animal" should look for bright and loud environment.

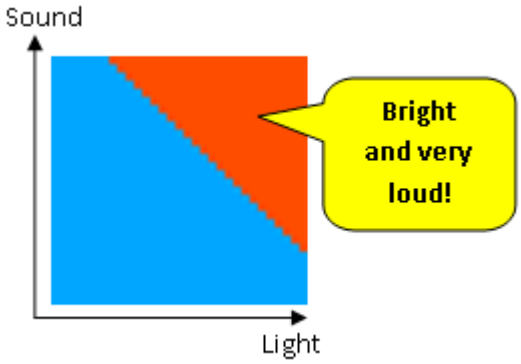


Fig. 8.12. A model of environment and preferences defined by the teacher

As You already know, at the very beginning the **Example 09** program examines the natural (built-in, inborn) preferences of the net. The second square You see on the screen (see fig. 8.13) should be regarded as a complete result of testing the "animal" before it starts to learn. The program just places the "animal" in every possible environment (checks all the points inside the square) and waits for its reaction. Each point is drawn red – if the reaction was positive, or blue – if the reaction was negative. The color scale is continuous to illustrate all the reactions between the two extreme stages defined. As we can see, at the beginning our "animal" "likes" dark places. It feels best in darkness and silence, but tolerates a little bit of light – generally the more sound it hears, the more light it accepts. Speaking not seriously, its favorite places are both the disco and the bedroom. Oh, what kind of an **animal** is that?! ☺ …

The network You will train on Your computer will surely have a different map of initial preferences, because – as You have already noticed – they are random. As the network's state desired by the teacher and the actual one are very much different (and it happens almost all the time experiments like this) an intensive training must take place.

Fig. 8.13. A presentation of the desired (by the teacher) and built-in properties of the net

You proceed to next steps of training by specifying the number of training steps to perform before the program pauses to show the decision map of the network. I suggest You watch the training effects relatively often in small nets, let's say every 10 steps. For larger networks, with several layers and many hidden neurons – much more training is needed to notice the results. So now it suffices to specify the lengths of subsequent training epochs, sit down and watch, compare and discover. That is just the best way to learn how neural nets work – better than studying loads of scientific books or articles.

I now suggest You analyze thoroughly some training stages that I have observed while preparing examples for this book – although You may obtain a bit different images on Your computer while trying to follow my experiments. Anyway, You will know what to expect.

A rectangle that I obtained after the first stage of training (see fig. 8.14) shows quite a similar image to that of a "newborn" network. This means that despite intensive training the net does not want to give up its original beliefs. Such an initial stubbornness and attachment to ones opinions is quite typical for neural network training and finds confirmation in many examples of people and animal learning.

Fig. 8.14. Net's state after the first stage of training

Next stage of training adds another square to our figure and shows an intermediate state of training. The network obviously starts to change its behavior, however it is far from the final solution – the boundary between positive and negative area is almost horizontal (fig. 8.15). At this moment the net thinks it knows the teacher's intentions: it should feel comfortable when it is loud. Unfortunately, it is not correct, so more adjustments are needed.

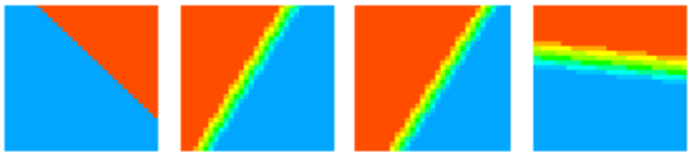


Fig. 8.15. Net's state after the second stage of training

Right after another portion of training, the network's behavior becomes more and more similar to the desired one, although the exact position of the decision boundary is not ideal yet (fig. 8.16).

Fig. 8.16. After three steps of training the net is close to success

You may consider it an outstanding performance – especially if You look at the short time it took to achieve such a result. Generally we should stop training at this stage. However, just for research, we continued training the net, assuming that our strict teacher aims for an absolute perfection and compels the net to adjust even after a really minor mistake.

The only effect of the next stage of training is a sudden crisis. The net's performance deteriorates – the decision boundary's position is worse than before (compare to fig. 8.17).

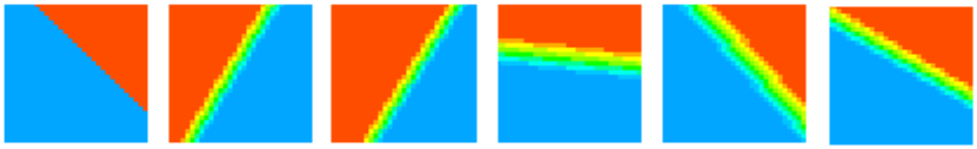


Fig. 8.17. Crisis in training, caused by a too strict teacher

Quite a long time is needed, taking many training steps, to recover and then achieve the accuracy imposed by the teacher. This effect is worth noticing. At this training stage when the net's knowledge is not yet formed and stable, excessive rigorousness of the teacher is almost always damaging, in extreme cases leading to a complete breakdown of the learner.

The whole process of learning this simple task by this simple net, after 12 training steps, is presented in fig. 8.18.

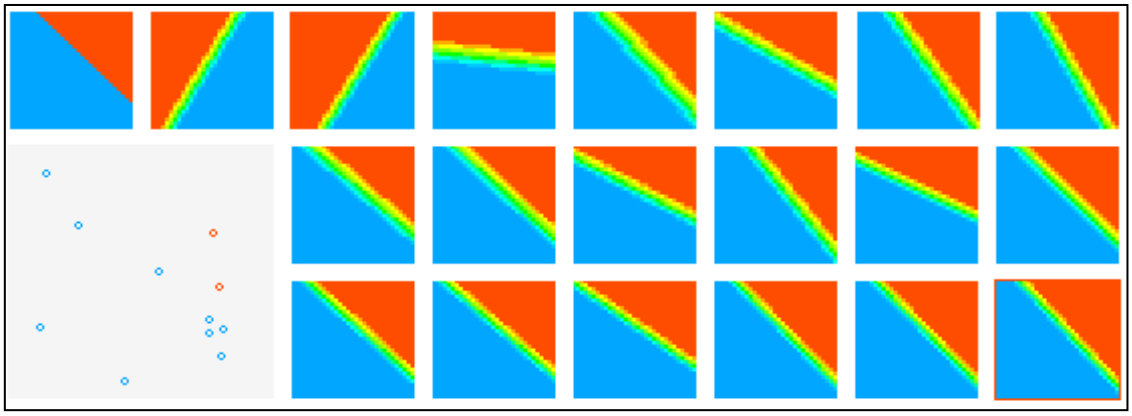


Fig. 8.18. Final result of training the net

Let us interrupt the program and go **Back** to the screen, on which we set the new parameters for the problem to be solved. After specifying the same net structure as before (a one-layer net) let us formulate a different (more difficult) task. Now we want our "animal" to be an enthusiast of the idea of "the golden mean" – let it discard all the extremes and feel comfortable only in a typical environment. It is easy to obtain by defining only one circle to be learned, as follows:

0,0,3.5

We shrink the other two circles to zeros:

0,0,0

0,0,0

The scheme of the ideal net's behavior is now presented in fig. 8.19.

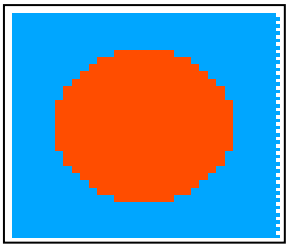


Fig. 8.19. The new task for the net

Observing the net's learning at fig. 8.20 (prepared in exactly the same way as fig. 8.18), we can see as the net struggles and continuously changes its behavior trying to omit the penalties imposed by the teacher – yet it has no chances.

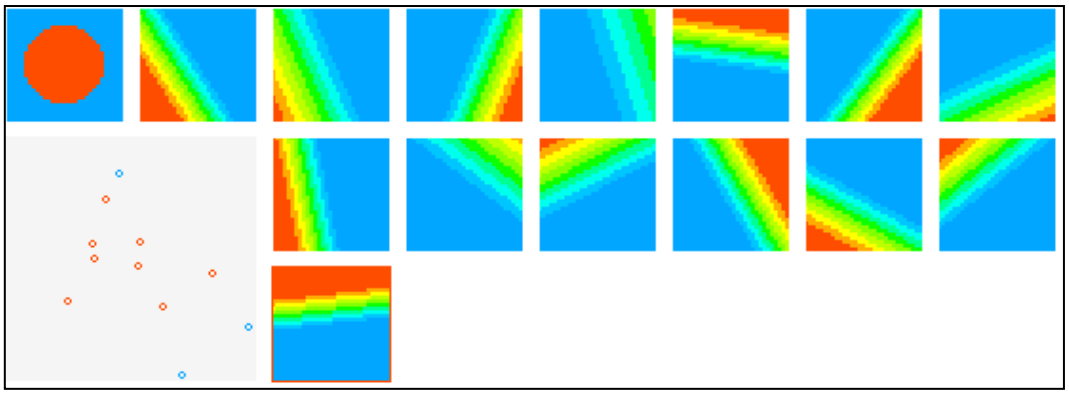


Fig. 8.20. Failure of training – reason: too simple structure of the net

Whatever parameters are chosen for the net and anyway the decision boundary between positive and negative reactions is set up – the net will always fall into a trap of accepting some region of extreme conditions.
E.g. many subsequent figures show that the "animal" tries to hide in the darkness, accepting the examples with a small amount of light – and that is what it will be punished for by the teacher in the next steps of training. Before I interrupted the experiment, the "animal" formulated another incorrect hypothesis – it started to avoid the regions of scary silence, thinking that the clue to success is to look for noise. Unfortunately, here it also happened to be wrong.

The reason for these failures is quite simple – the modeled neural net was too primitive to learn such a sophisticated behavior as "choosing the golden mean". The net we used in our experiment was able to understand only one way of input signals selection – linear discrimination, which for the second problem turned out to be too simple and primitive.

Therefore the next experiment will engage a larger and more complicated network for the same task. When setting the net's structure, we may specify the parameters as shown in fig. 8.21.

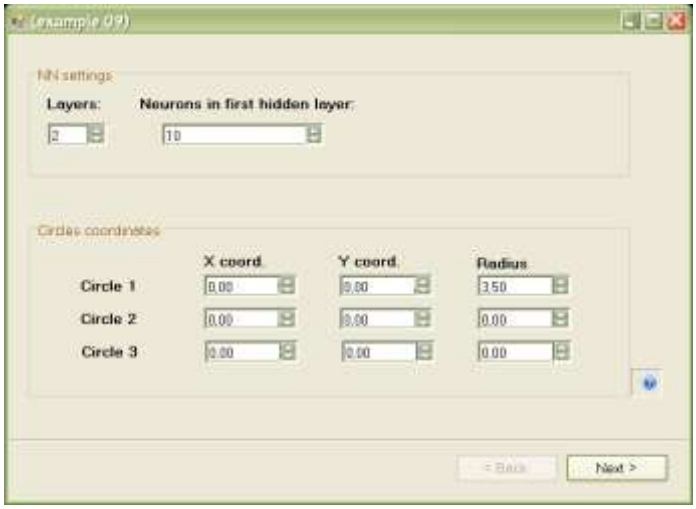


Fig. 8.21. This way we set up a "more intelligent" net

It is worth noticing that after declaring more than one layer in the net – our program asked for the number of neurons in it. The more layers You need, the more answers You give.

Fig. 8.22. An example of the initial state of a more complex net

As You know, the more layers and the more neurons are in the net, the wider is its "intellectual potential". Such a net can really make better use of its "in-born intelligence". Let us watch now how it learns. Its initial distribution of color areas, shown in fig. 8.22, indicates that the net is initially – by itself – enthusiastic about everything. The modeled "animal" feels comfortable in almost all conditions, although it loves dark and quiet corners.

Fig. 8.23. After a few training steps

First examples of training show the net that the world is not so beautiful, life is not always sweet, but sometimes also bitter. The net reacts in a typical way – at the beginning it cultivates its initial prejudices, it is definitely less positive about dark places after the first stage of training, but it is still far from the final success (fig. 8.23).

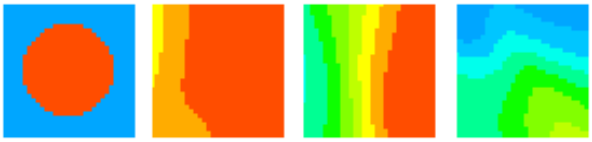


Fig. 8.24. After some more training steps

Further training leads our "animal" to a series of disappointments. It receives more punishment for being too much trusting and enthusiastic, and so it retreats to more suspicious attitude towards the whole untrustworthy world. We can see that the initial enthusiasm and favor to everything and everyone diminishes into a slight fondness for quiet and sunny places (the small dot of yellow at the bottom and to the right of the last square on fig. 8.24).

Fig. 8.25. A crisis of total negation during training

But this attitude also meets the teacher's disapproval, so after the next training stage the net goes into a complete stagnation – it rejects **anything** at all. Such a state of absolute breakdown and discouragement is quite typical for neural nets training and usually it precedes an attempt to construct a positive representation of the knowledge desired by the teacher.

Fig. 8.26. A positive wave of optimism

In fact our net really tries to create an image like this, as we can see in fig. 8.26 –only a few positive examples shown to the net by the teacher are enough for it to fall into a phase of enthusiasm and optimism in which it approves all the environments except those very loud and not too bright. Total darkness or intensive light reduce its tolerance for loudness.

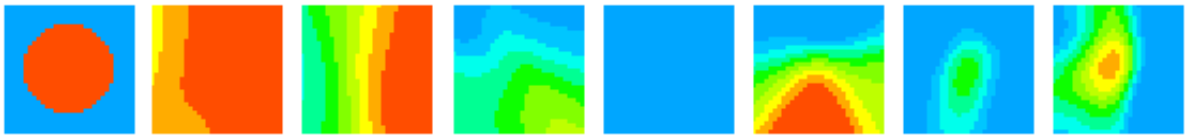


Fig. 8.27. Formation of the proper hypothesis

Of course, the next training stage must suppress these enthusiastic hopes, so the net backs up again into a region of rejection and frustration – leaving a small trace of only those positive memories that have not caused severe punishment (the green “tongue” in the figure illustrating the net’s state after 5 training steps). Not later than in the next stage this small area will become a seed of correct hypothesis (the red spot near the middle of the volatility region – fig. 8.27).

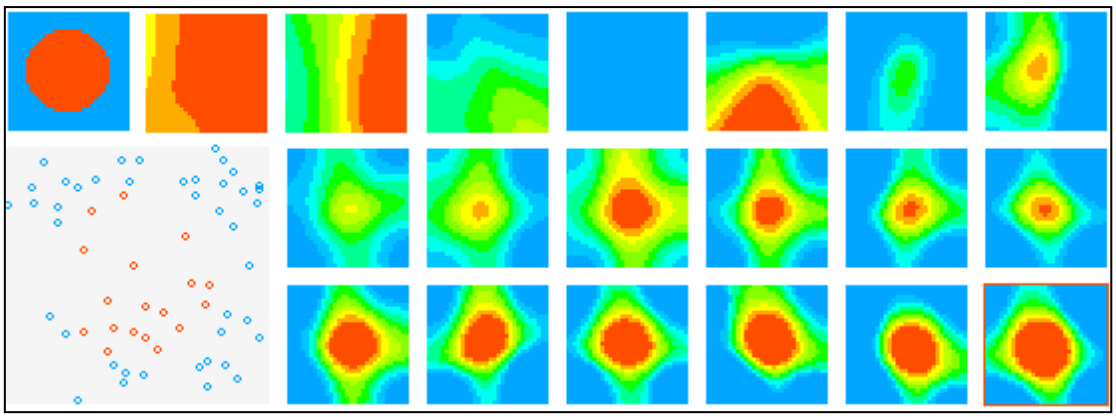


Fig. 8.28. The whole process of learning a more complex task

After that, training is only about controlling another rush of enthusiasm and reducing the area of positive reactions to a reasonable size. When I interrupted training (after 15 stages) the net was able to quite appropriately follow the teacher in this signal classification task so that further improvement of its performance was not necessary (fig. 8.28)

As we can see – the system with larger and more efficient “brain” (more than ten times more neurons! and more powerful structure of connections between them) turned out to be able to learn the behavior that the smaller “creature” could not manage to learn.

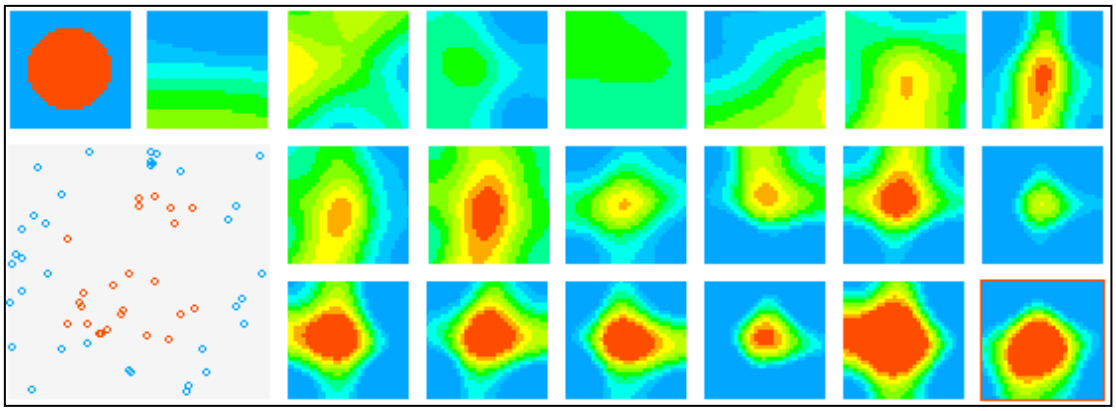


Fig. 8.29. Training process of another net

Another interesting experiment is presented in fig. 8.29. It illustrates how the same ability (of deducting that the teacher requires accepting medium light and sound conditions and rejecting the extremes) is learned by another instance of the net with the same structure as the one analyzed in fig. 8.28

The training process of the second net is presented in fig. 8.29. Despite the fact that the net has exactly the same structure as the first one, it is rather melancholic at the beginning of training. In fact it does not like anything – the figure is dominated by cool green and blue colors. Initial training makes it a little more interested in “disco” conditions (a yellow spot in the rectangle illustrating the first stage of training), but several consecutive failures throw the net into an area of total (yet modest) rejection. At the fourth training stage the net tries to put forward a hypothesis that the teacher wants it to approve bright and medium loud places. At the next stage the net’s guess aims at medium and average conditions to be accepted. Then from one training step to another this hypothesis grows stronger and clearer. This is illustrated by the red dot of complete acceptance. With each training step the dot’s shape matches more exactly the acceptance area given by the teacher. Green and yellow areas of confusion and uncertainness vanish replaced by more and more exact division into utterly good and bad conditions. Finally the “melancholic” net learns to act exactly the same way as the “enthusiastic” net, what proves that the built-in predispositions are not principal – strong, determined training always achieves its goal, however the way of achieving it varies much for different starting points.

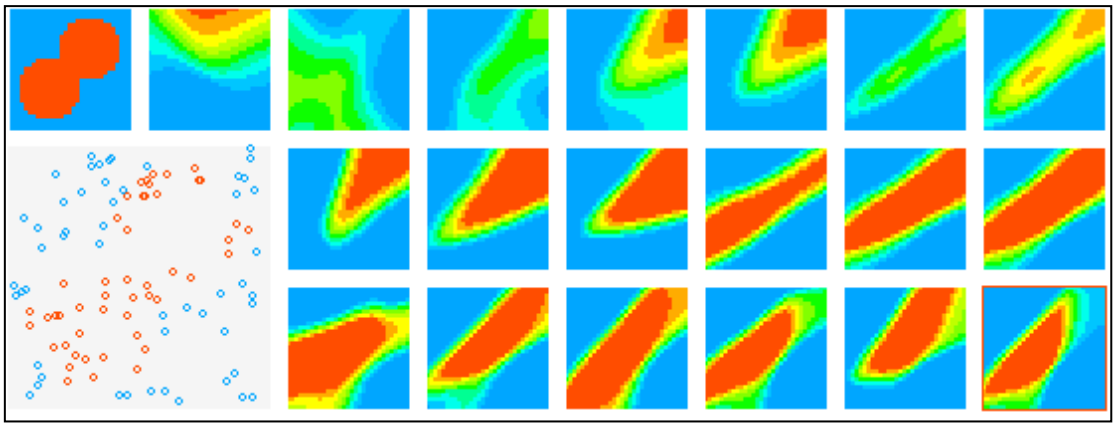


Fig. 8.30. An example of a problem for two-layer net

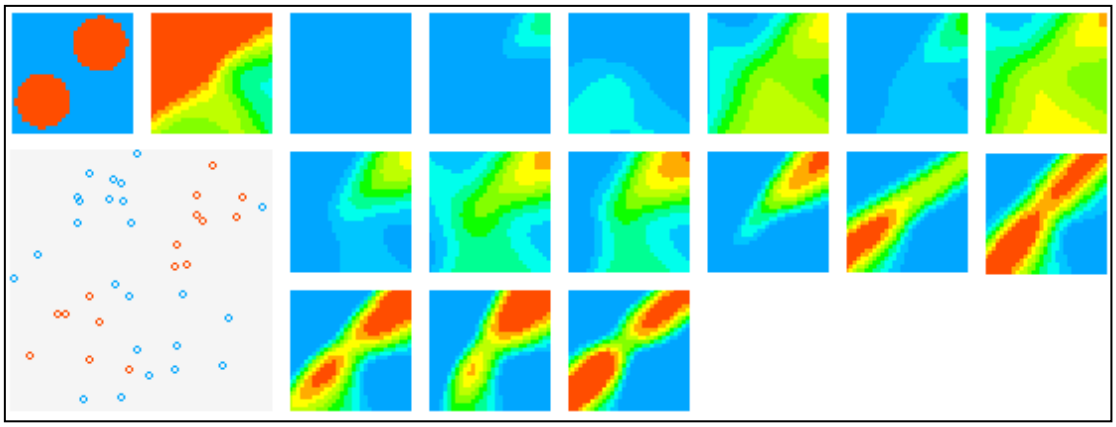


Fig. 8.31. An example of a problem not every two-layer net is able to solve

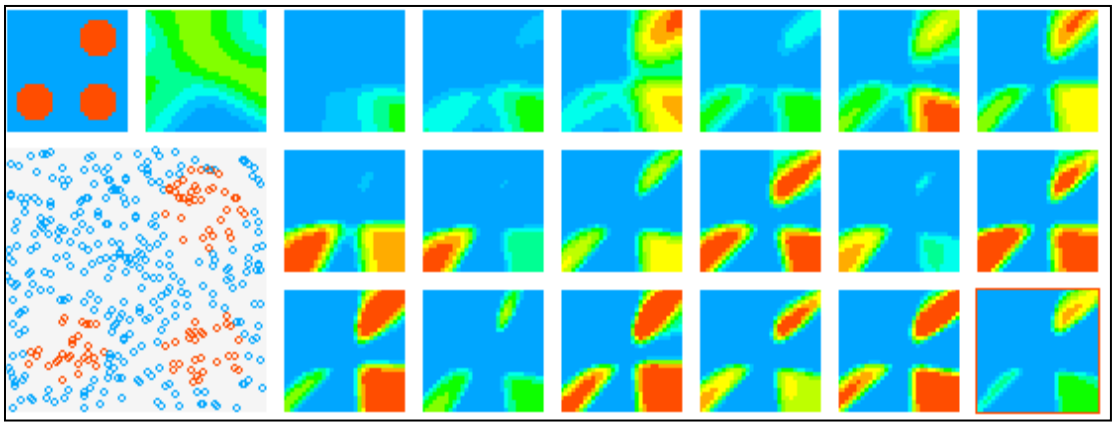


Fig. 8.32. A very difficult problem only for a three-layer net

You my perform many kinds of experiments using our program. Figures 8.30 - 8.32 show the record of a few of them. Analyzing these figures, reconstructing them on Your computer, conducting Your own experiments – may be fascinating world for many discoverers. It is really worth exploring!

One conclusion just to finish this chapter. Please compare figures 8.30 – 8.32 with figure 8.20. You may notice how the behavior of a well "intellectually equipped" net varies from the behavior of a simple net that was absolutely unable to learn more sophisticated rules. Let us notice that this neural "moron" could not learn simple behaviors, yet its reactions were very strong and categorical. The world was just black and white for this net. One object was absolutely good, another one was definitely wrong. If during the experiment the net's experience did not confirm these harsh rules, they were replaced by another – opposite – opinions, far more strict and rigid, usually also incorrect. As for the nets with more "intellectual power", although they were able to model more subtle divisions, they were much slower in reaching the goal and stayed longer in many such stages as: hesitation, breakdown and "hamletizing". Yet it were these neurotic, hesitating, excessively intelligent individuals that reached the goal of understanding the rules imposed by the teacher and adjusting to them, while the tough moron was turning round and round and could not make up any rational conclusion…

Notice how interesting and diverse behaviors of neural nets we were able to observe due to one simple program. The computer running it did not resemble a dull and limited machine, it did not act monotonously and repeatably – on the contrary, it demonstrated some individual characteristics, mood changing, a variety of talents… Doesn't it remind You of anything?

8.6. What else can we observe in our net?

(Translation by Piotr Ciskowski; piotr.ciskowski@pwr.wroc.pl)

I could end now, leaving You with the program to discover more interesting aspects of training, but I will try to suggest You, what sample shapes are worth investigating, before You gain some experience. My experiments showed that interesting training process may occur if You use a "snowman" made of three circles:

0,03
4,0,1
0,4,1

The task is complicated enough so that the net must work hard to discover the actual rule of recognition (the net for this task should be 3 layered, and the number of neurons in hidden layers should be high – e.g. 17 neurons in the first and 5 neurons in the second hidden layer). At the same time this task is compact enough to be displayed well on the screen and so the results are easy to observe. I will not show You my results for these examples, as I do not want to spoil Your personal pleasure of discovering them yourself. I will only show You what results You may obtain if You decide to explore a bit more the properties of neural nets and their sensitiveness to some parameters that affect the training.

As You already know, before the program executes the number of training steps You chose, You may also specify some parameters of training that influence its speed. The first one, defined as *Alpha* is known as the *learning rate*. The greater the learning rate the more intense the training process is. Sometimes high *Alpha* gives good results, but other times may worsen them – it is worth trying on Your own. The second parameter, denoted here as *Eta* is known as *momentum*. It makes learning a little "conservative" – the net does not forget the previous directions of weight changes while performing the current training step. Again – in some cases higher values of *Eta* help in learning, other times they slow it down.

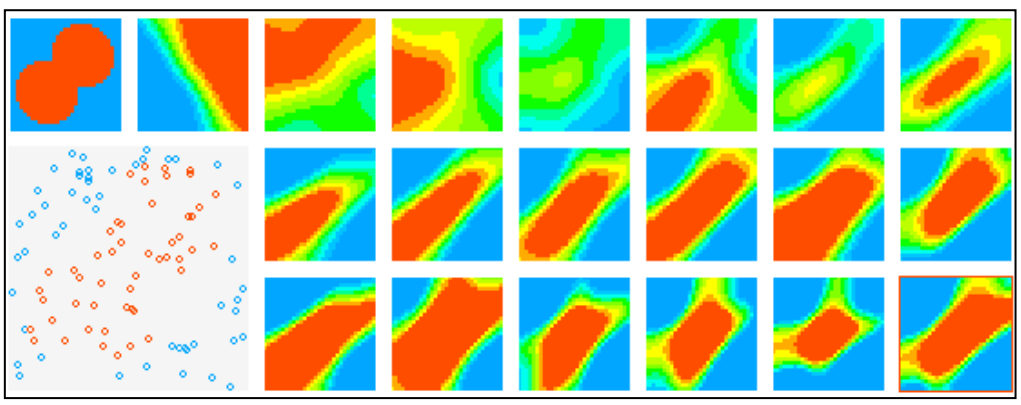


Fig. 8.33. The default value of learning rate leads to quiet and steady but slow training

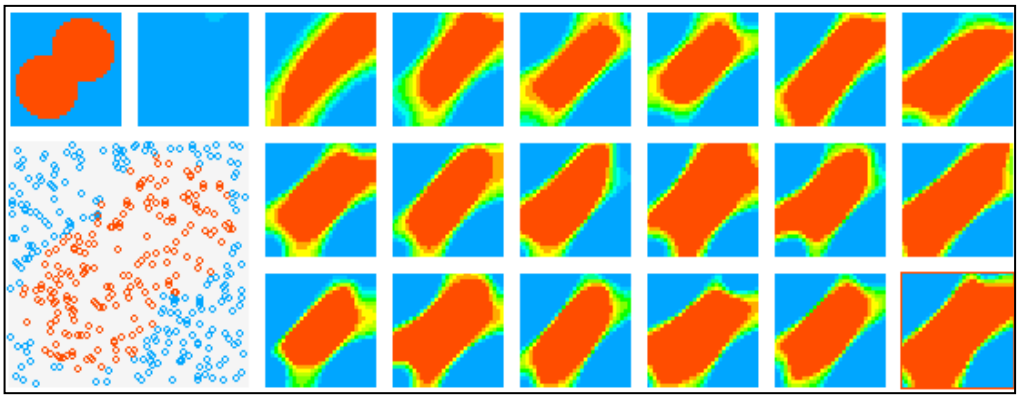


Fig. 8.34. Higher value of learning rate sometimes gives faster training

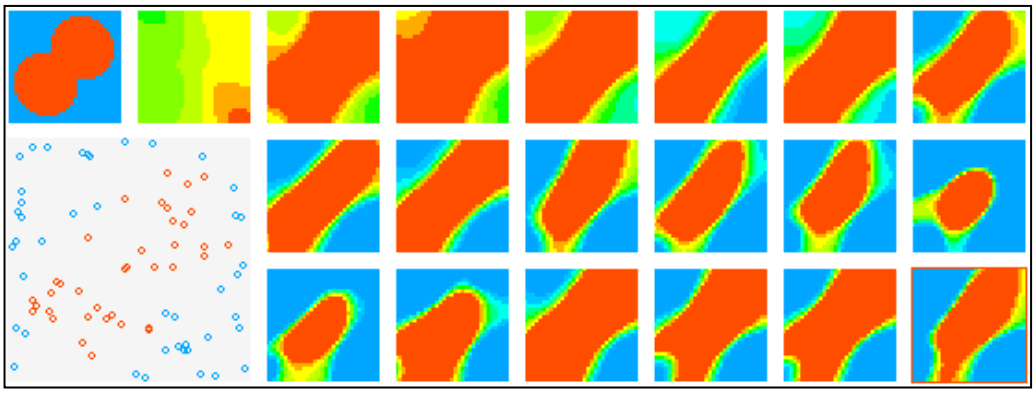


Fig. 8.35. Too high learning rate ends up in oscillations
– the net in turns comes close to the desired solution, then goes away

Both mentioned coefficients are constantly accessible in the program window of **Example 09**, so You may change them during training to observe interesting results. You may start with modifying them only once before learning starts. You should notice how training speeds up with higher values of *Alpha*. E.g. fig. 8.33 presents the training process for some task with standard values, while fig. 8.34 shows training with higher *Alpha*.

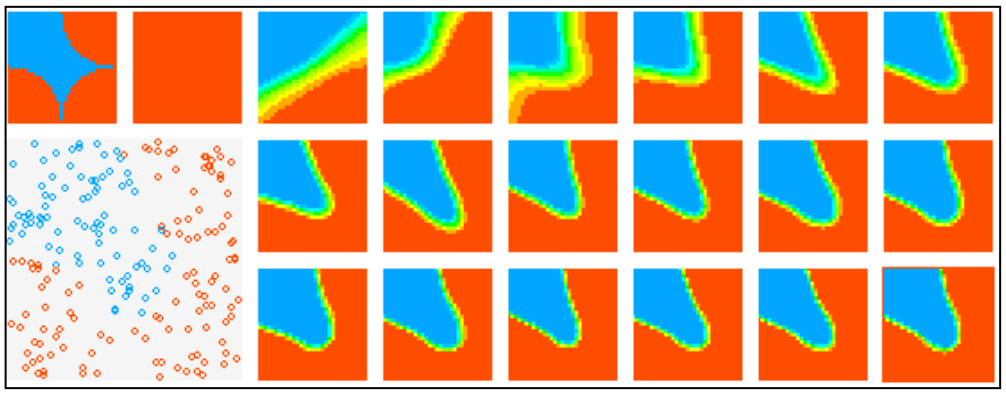


Fig. 8.36. The regions learned by the net do not match the desired ones
if we use a two-layer net

Too high *Alpha* is not good, however. Signs of instability occur, as it is perfectly illustrated in fig. 8.35. The way to suppress the oscillations, which are always the result of a too high learning rate, is turning

up the *Eta* coefficient, that is *momentum*. If You wish, You may analyze the influence of changing *momentum* on training and notice its stabilizing power when training goes out of control.

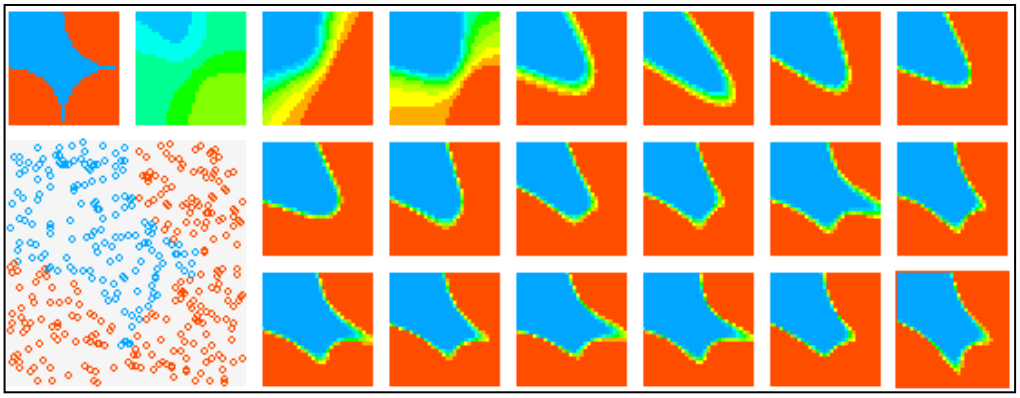


Fig. 8.37. Good solution found quickly by a three-layer net

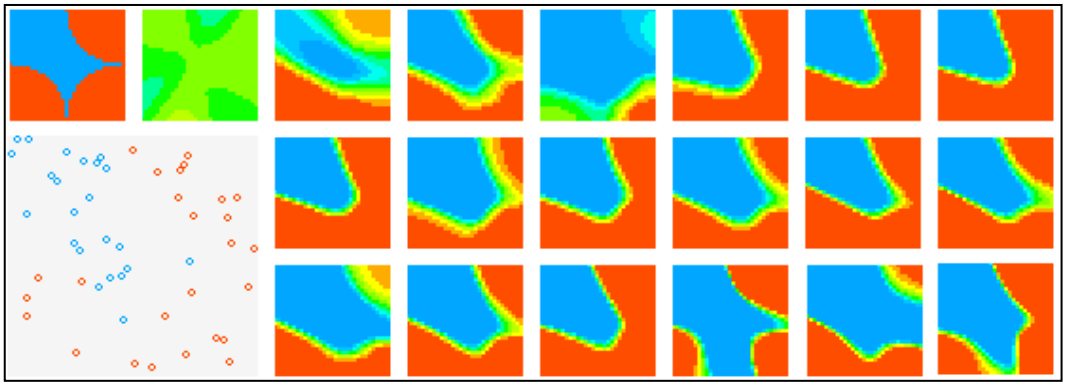


Fig. 8.38. Signs of instability during learning of a three layer net

Another possible area of interesting research is the dependence of net's behavior on its structure. To analyze that You must choose a really tough problem to be learned. A good one is presented in fig. 8.4 and 8.5. The difficulty here is the necessity of fitting into narrow and tight bays defined by the blue area of negative answers. In order to fit to each of these slots a very precise choice of parameters is needed and that is really a big challenge for the net.

A three layered net is able to distinguish all the subtle parts of the problem (fig. 8.37), although it is sometimes difficult to keep the training stable, as it happens that the net – even being very close to the solution – moves away of it due to the backpropagation of errors and searches for solution in another, incorrect, area where it cannot find it (fig. 8.38)

Now You may use your imagination and freely define the regions where the net should react positively and negatively. You may run tens of experiments and obtain hundreds of observations. Although I am warning You – think of your precious time, as training a large net may take long time especially in its initial stages. What You need is either a very fast computer or some patience. I am not forcing You what to choose, but just suggest that the patience is cheaper…

8.7. Questions and exercises

(Translation by Piotr Ciskowski; piotr.ciskowski@pwr.wroc.pl)

1. Does image recognition only to refer to visual images (e.g. from a digital camera) or does it have a wider meaning?

2. How do we input the image data (and information about other objects to be recognized) to a classifying neural net?

3. How do neural classifiers usually present the outcomes of their work, that is the information on what was recognized? What consequences does it have on the usual structure of their output layer?

4. Try to confront the observations we made in this chapter on training the nets for several generated problems – with theoretical information on how multi-layer networks (with different number of layers) work, presented in chapter 6. Notice the higher capabilities of the nets presented in this chapter, resulting from the use of neurons that are able to produce continuous output signals, not only 0 and 1 (what refers to the blue and red color in the figures given by the program).

5. Think about the relation between the number of hidden neurons in a complex multi-layer network and the number of separate areas in the input space, for which the net is able to learn opposite decisions (0 and 1).

6. An example of relation between the number of hidden neurons and the efficiency of the net in performing several (trained) tasks in illustrated in fig. 8.39. The left side of the plot is quite easy to interpret: a net with a small number of neurons is not "intelligent" enough o solve the problem and adding more neurons improves its performance. How should we interpret the fact (which has also been proved many times in neural nets) that too large number of hidden neurons – by irony – worsens the performance of the net?

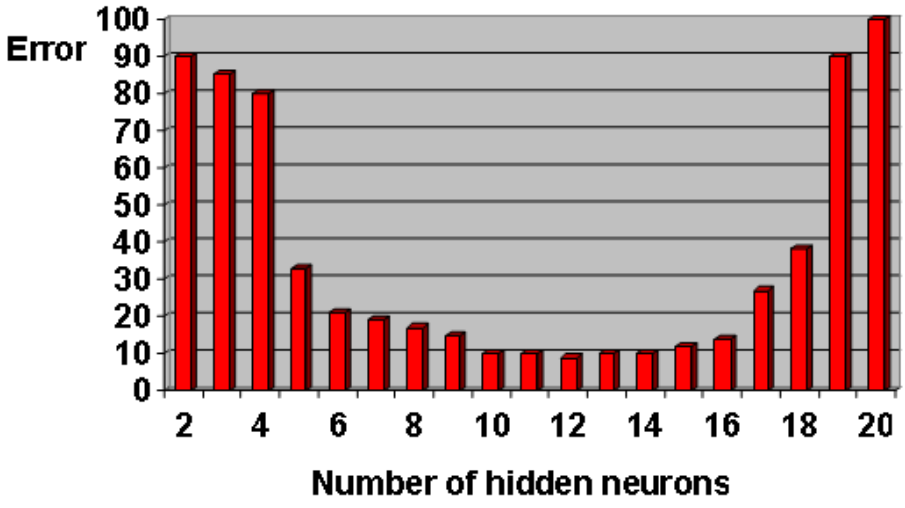


Fig. 8.39. See exercise 6

7. *Learning rate* is the measure of intensity of weight adjustments applied to the net after detecting wrong operation. The informal interpretation for the learning rate is that its high values represent a "strict and demanding" teacher, while small values correspond to a "gentle and tolerant" one. Try to justify the fact that the best results of the training process are observed for moderate values of learning rate (neither too small, not too high). Try to confirm that experimenting with our program. Draw a plot of leaning rate vs. training performance after some given number of epochs (e.g. after 500, 1000 and 5000 steps) .

8. Many training methods are known, in which the learning rate is not constant throughout the whole process. What do You think, in what conditions should it be increased and when should it be reduced?

9. Some training theories suggest that the learning rate should be small at the beginning, when the net makes a lot of major mistakes. If we apply strong changes to the weights, according to these large errors – the net would "go into convulsions", avoiding some mistakes, but making other ones (for other training examples). So at the beginning the net needs a "gentle and mild nursery school teacher". Then, as the net gains more and more knowledge, we may increase the learning rate, because the net make less mistakes, and they are less significant, so for successful teaching, "tightening the screw" is needed. At the end of training, when the net has gathered a lot of knowledge – again a small learning rate is advisable, so that incidental mistakes (e.g. those associated with a few hard and unusual objects in the training set) do not spoil the whole effect (as an improvement in one point always results in – even smallest – worsening in other points). Try to confirm this theory or negate it with your experiments.

10. The effects of a too large value of the learning rate (putting at risk the stability of training) may be compensated by increasing the *momentum* coefficient, which in turn is the measure of *conservatism* of training. Design and conduct some experiments in order to draw a plot showing what minimal values of *momentum* are needed for different values of learning rate, to guarantee fast and stable training process.

11. **For eagles:** Modify our program so that it is possible to define the regions for different classes (areas where the "animal" should feel comfortable and uncomfortable) with the use of a graphical editor. Using this tool, try to build a net that will learn an especially difficult problem, in which these regions are defined as two spirals (fig. 8.40).

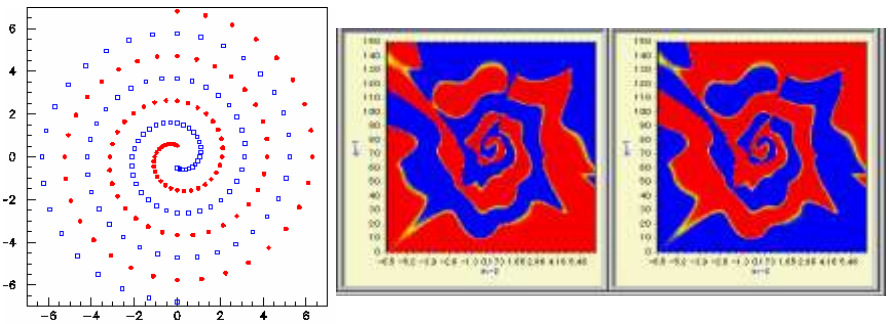


Fig. 8.40. The problem of "two spirals": training set (left) and two possible solutions (right) - see exercise 11

12. **For eagles:** Design and make a model of an “animal” with more receptors (“senses” to observe various features of the environment) and more sophisticated in its actions (e.g. able to move in the “environment” in order to look for conditions it likes, or to search for some objects, e.g. food) . Observing such a neural net, as a “brain” of an animal, may be a fascinating intellectual adventure!

9. Self-learning neural networks

9.1. What does the self-learning of neural networks rely on?

Not translated yet …

9.2. What is the way that long self-learning of a network proceeds?

Not translated yet …

9.3. Can the progress of self-learning be considered as growing wise of a network?

Not translated yet …

9.4. What is also noteworthy during the self-learning process of network?

Not translated yet …

9.5. Dreams and imaginations arising during the self-learning of neural networks.

(Translation by Ryszard Tadeusiewicz, rtad@agh.edu.pl)

Once you observe how notions spontaneously discovered by the network are formed and specified - you can pass on to more subtle phenomena. You probably notice (particularly when practising examples with a large number of neurons), that beside the main process of formation of neurone clusters recognising main objects introduced to the system, we also get entire tracks of neurones. These neurones are spontaneously aspiring to detect and recognise input objects, having the properties shared by real-life objects (Fig. 9.24). During the further process of self-learning these detectors of hybrids get absorbed by real centres signalling the presence of real-life objects. However this stage of spontaneous fantasizing about possible, though not existing entities is repeated in self-learning networks with an astounding regularity.

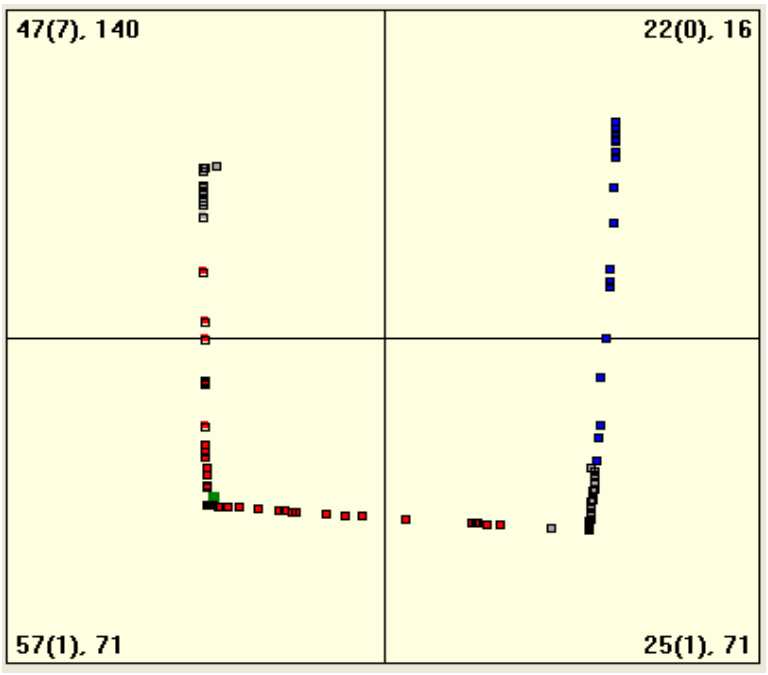


Fig. 9.24 Appearance of parameter localization for neurons (detectors of hybrids) ready to detect objects having the properties shared by real objects

What does it mean, really ? A network which is repeatedly shown a fish and a woman will be able to recognize both the woman and the fish. At appropriate points of space of input signals there will be neurons which correctly recognise input images of a woman and a fish. There will be also neurons which will recognise (or rather, which will be ready to recognise) entities displaying the features both of the fish (tail) and of the woman (the head and shoulders). During the self-learning process such creatures weren't shown – and yet the neural network prepared neurons for detecting them. It is reasonable to suppose that in a certain sense the network itself would imagine them.

Following the similar principle, the same network which developed an ability to recognize bird species in a group of neurons is able to associate bird features with the characteristics of the woman (the head, hands, the dress), thus creating … an image of an angel (see Fig. 9.25).

Having repeated numerous experiments, we become convinced that every young (not fully taught) neural network will have a tendency to invent various not existing objects. If new experiments regularly prove that both women and fish do exist and there are no creatures sharing the properties of both women and fish- such as legendary mermaids, the relevant neurons will change their specialisation.

These neurons will proceed to recognise real life creatures instead of imagined phantasmagorias. We notice how unwilling they are to do so during the experiments! But if such a hybrid creature should exist (a mouse with wings = bat) – the previous experience with “component’ objects will definitely facilitate the detection and classification tasks.

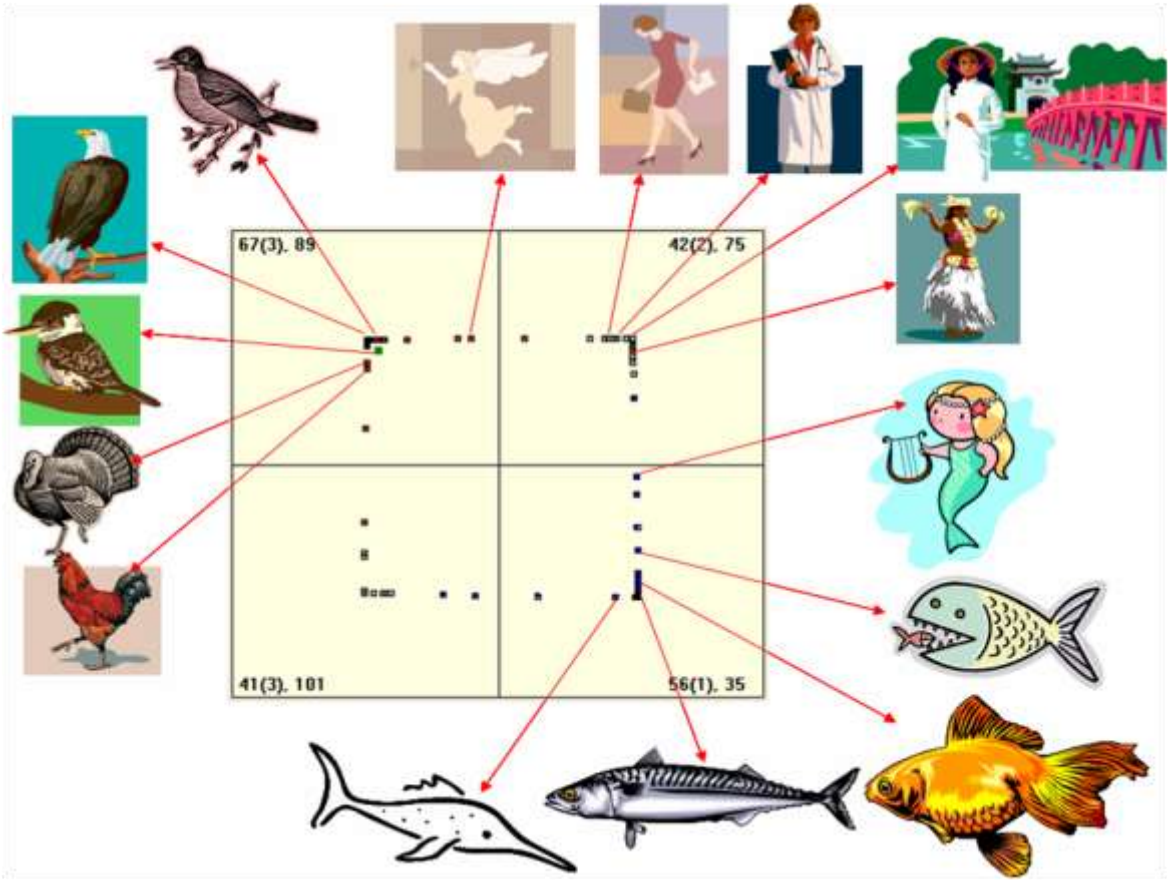


Fig. 9.25 Representatives of real and imaginary objects in self-learning the neural network

It appears that this ability to invent not existing objects, combining elements of sometimes distant impressions and experiences is the property displayed only by networks in the early stage of self-learning (also referred to as young networks). Perhaps that partly explains the well- known tendency among young children is enjoy fairy tales and legends, which seem boring or even annoying to adults and more experienced people? Perhaps that also explains the phenomenon of poetry, full of fantasy and youthful imaginations in young civilisations (ancient Greece) and the blunt pragmatism of old and stiff societies in the early 21st century? Maybe when observing (Fig. 9.26) feeble chains of neurons emerging in the course of self-learning, which, despite obvious facts, are trying hard to maintain the non-existing links between the detected and specifically localised phenomena, you are actually watching the neural mechanism of formation of brilliant associations, distant analogies and surprising metaphors in certain minds?

Perhaps such small bridges, spontaneously coming into existence, but then collapsing under pressure of new facts, products of associations and phantasmagorical dreams, fantastic combined features of existing objects, created by the very neural mechanisms in order to produce something which may not exist but is beautiful and exciting, are the very essence of poetry, art as well as creative science? And now perhaps, you are closer to understand the well-known fact that poets (and real scholars) are a little bit like children .....

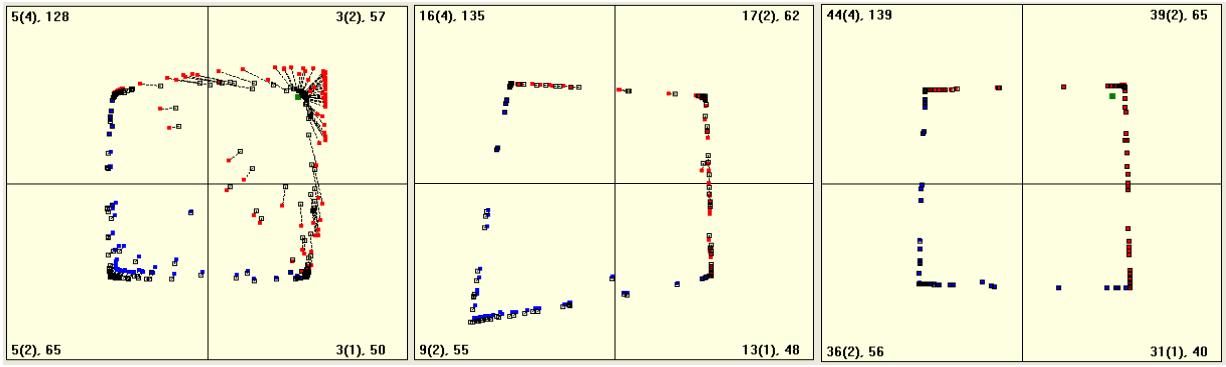


Fig. 9.26 Voluntary coming into existence and destroying associations in the process of self-learning

We notice during some experiments that some (usually single) neurons will remain outside the areas concentrated near the spots where input objects shall turn up. Actually, they are of little practical significance since the vast majority of neurons shall be meekly taught and ready to subject to more and more precise specialization. The majority of neurons is aiming at perfection in handling the task, involving detection and signalling the objects that regularly turn up in the real world. And yet, these neurons that are out of touch with reality will stay and retain (Fig. 9.27) their images of winged snakes or multi-head dragons - as dreams returning when you sleep...

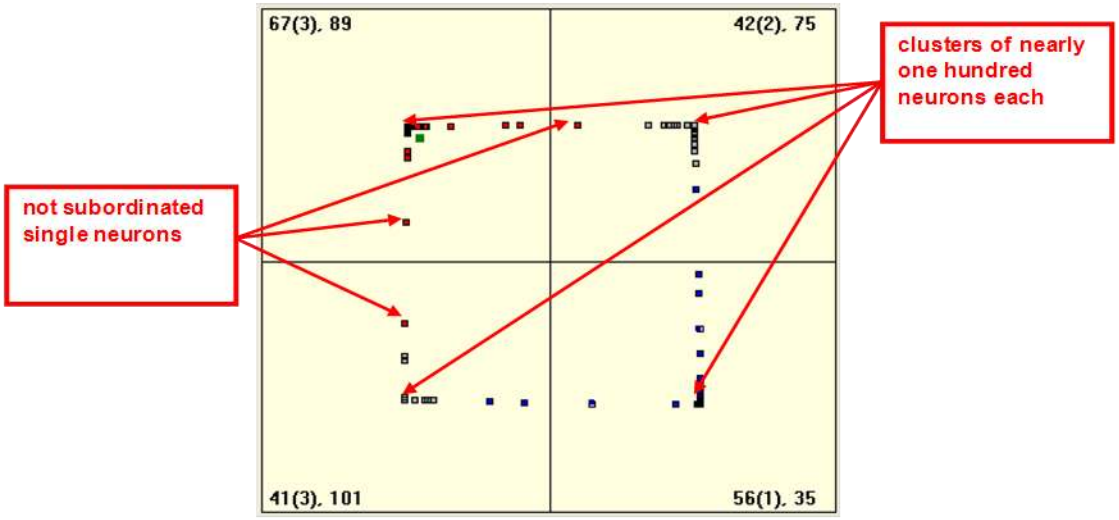


Fig. 9.27 Appearance of singles neurons outside global centres at a very advanced process of self-learning (after closing two hundred steps)

Let us now return to further experiments. If you watch carefully the initial stages of self-learning, then in some (though rare!) cases you still notice another interesting phenomenon. After an object is shown at a certain point in the space of input signals - some neurons (very few in general, but hence more worthy of our attention) are changing their initial 'decent' position. These neurons reach the new location, distant from their initial position, however remaining in the same direction as the input object, though much farther from the origins of the coordinate system. As I said before, this phenomenon is rather rare because it strongly depends on the initial distribution of weight coefficients (well, not everyone is born a poet...) but if you patiently repeat your experiments, you are bound to notice it sooner or later (Fig. 10.28). As a rule, such rebellious neurons land outside the

borders of the limiting frame and therefore it is possible to observe them mainly because lines will appear with no empty square at the end, which suggests that it lies beyond the display area.

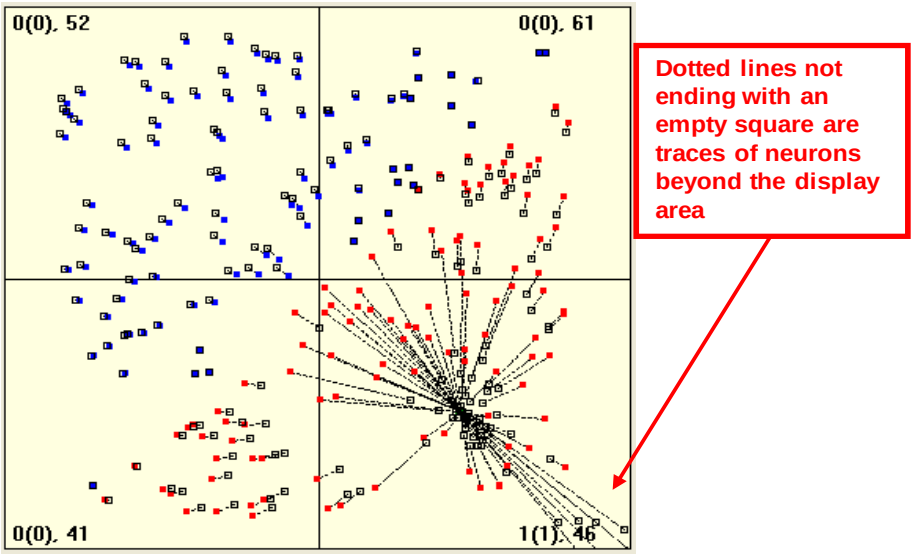


Fig. 9.28 Effect of neurons escape in the initial stage of the process of self-learning

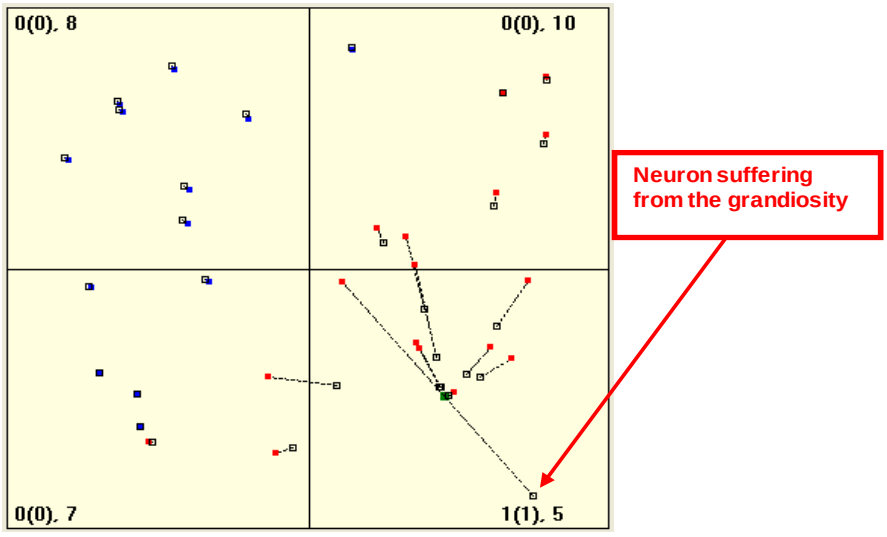


Fig. 9.29 Effect of grandiosity in a little lower scale, than in the 9.28 picture, but also noticeable

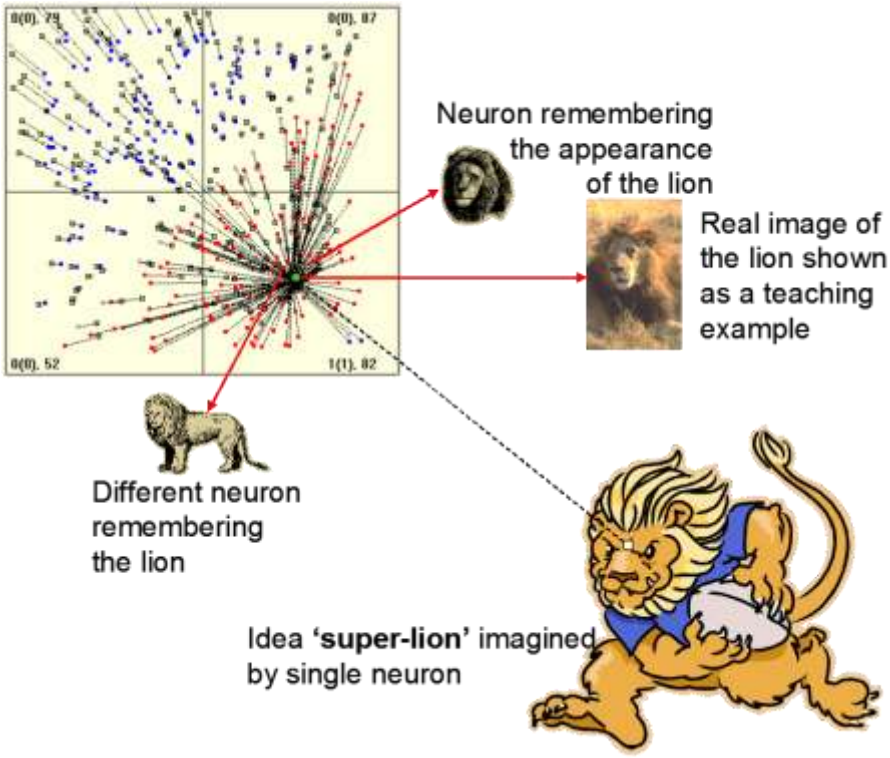


Fig. 9.30 Interpretation of the effect of 'grandiosity'

Neurons running far at the beginning of the self-learning process will be pulled back in consecutive stages of the teaching process to the area associated with position of the presented standard and, finally, they will all find their way back to this area. We wouldn't worry about them at all if it weren't not for the fact that they illustrate a well- known psychological effect – the effect of grandiosity (Fig. 9.29). Indeed, such a neuron jumping out beyond the display area becomes the standard (an internal depiction) of the object which has roughly the same features as the model object perceived at the beginning of the teaching process - yet with all its features magnified. Such a neural Gulliver. If a lion is shown, the image produced after enlarging its perceivable features becomes that of a real monster. Thus depicted lion is much larger in size, has huge fangs and claws, more ruffled mane, its roar is more dangerous, its breadth malodorous. All features are those of a lion, but somehow larger. 9.30 fig.. Such exaggerated ideas do not usually last long as they are reviewed and verified by life long experience. But for the naive and quick-tempered Youth just entering the world of new experiences, everything seems huge. We remember that the first Girl is always The Best! I wonder if you noticed and appreciated the fact that the neural network invented by me and examined by you will do exactly the same?!

The saddest thing occurs at the end of the self-learning process. The majority of neurons form tight clusters gathered near the spots where input objects will turn up (everybody's already chosen their master and are ready to serve him...). And then, sometimes from beyond the display area, some lonely, ill-adjusted points appear and, despite the resistance they put, they are relentlessly pulled back to the areas where real appearing objects are recognisable (Fig. 9.31). Their views are different from those of the majority. What is even worse, everyday life shows that their views are wrong. And yet they prefer to hold on to a beautiful fairy tale instead of recognising laws of real world where different value still count...

Yeah, but let us leave it and come back to our neural network. After a certain time, rebellious neurons are absorbed by the "mass: forced to adapt to the majority, imported to the right path. However, there is something beautiful in their rebellion. It is worthwhile to mention that when something new and really unexpected appears in the world that provides the neural network with input impressions, those ill-adapted dreamers will get a great chance to win.

A crowd of perfectly adapted conformists won't succeed in this new situation. A pity that it is such a rare occurrence this way - even in neural networks!

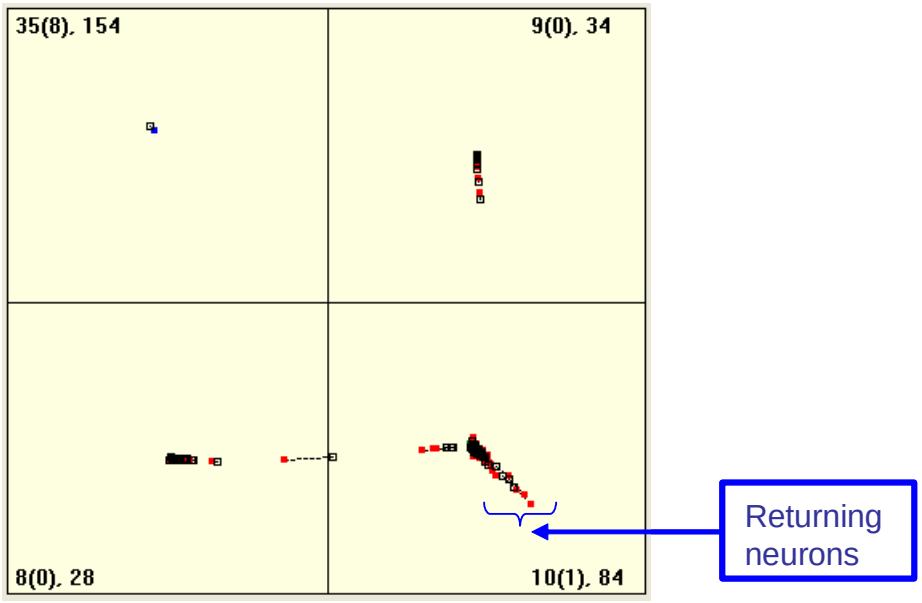


Fig. 9.31 Pulling back the neurons which escaped in the initial stage of the process of teaching, creating fictional representations of data described higher under the name of grandiosity

9.6. Remembering and forgetting

(Translation by Weronika Łabaj, weronika.labaj@googlemail.com)

In this chapter we concentrate on processes such as storing information and gaining knowledge in self-learning networks, especially on how are they done. However, you can name another process occurring on a daily basis in our lives - I mean **forgetting.** It is burdensome (especially before an important exam) but is essential in terms of biology. Our living environment is changing constantly, so what we learned and what was working at some stage of our lives turns out to be outdated (even harmful) as we are gaining new experience. Because of that (among other things) we are forced to constantly gain new skills and abilities at the same time **forgetting** previous ones. Otherwise they could be easily mixed or confused.

Use your neural network simulation to observe and analyze that phenomena. It turns out that even having only a few self-learning neurons you can notice that new objects appearing in time of simulation "divert" neurons from other, not so common classes of objects you could consider well learned already. In extreme cases **new** objects can sort of "kidnap" neurons - neurons which

previously recognized patterns quite well, neurons which were painstakingly taught patterns that don't appear any more. See figure 9.32. In this situation neural network quite well recognizes all four patterns that were presented to it. Note that pattern no. 1 has a strongest representation (1st quarter). At this stage self-learning process is disrupted (on purpose) – all objects are shown but from the 1st class. In no time we get to the stage of "kidnapping". Neurons recognizing objects of class 1 change their specialization in a hurry. They start specializing in recognition of other classes of objects (in our example of class 4 – see Fig. 9.33. left). That way the previously best-recognized class is being forgotten. Not completely though. Even a very long learning process leaves a trace in memory. It reminds of existence of that class (Fig. 9.33. right). But this trace is really weak (only one neuron) and strongly distorted (position of this neuron was significantly altered).

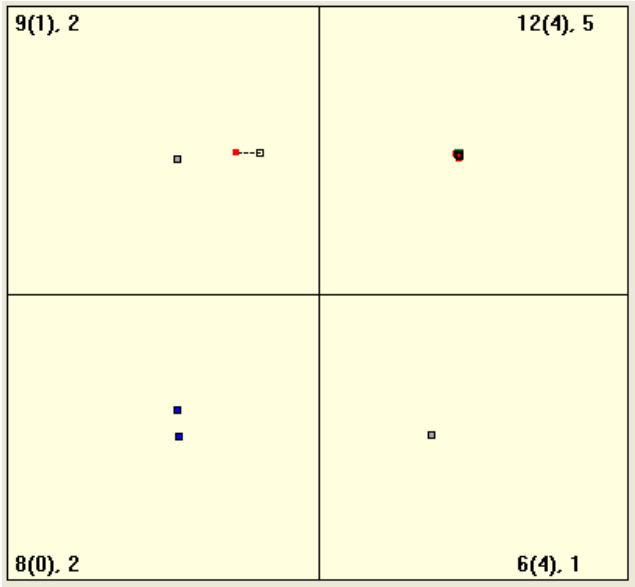


Fig. 9.32. First stage of forgetting process occurring during self-learning process. At the moment network posses yet "old knowledge" and best recognized class is class number 1.

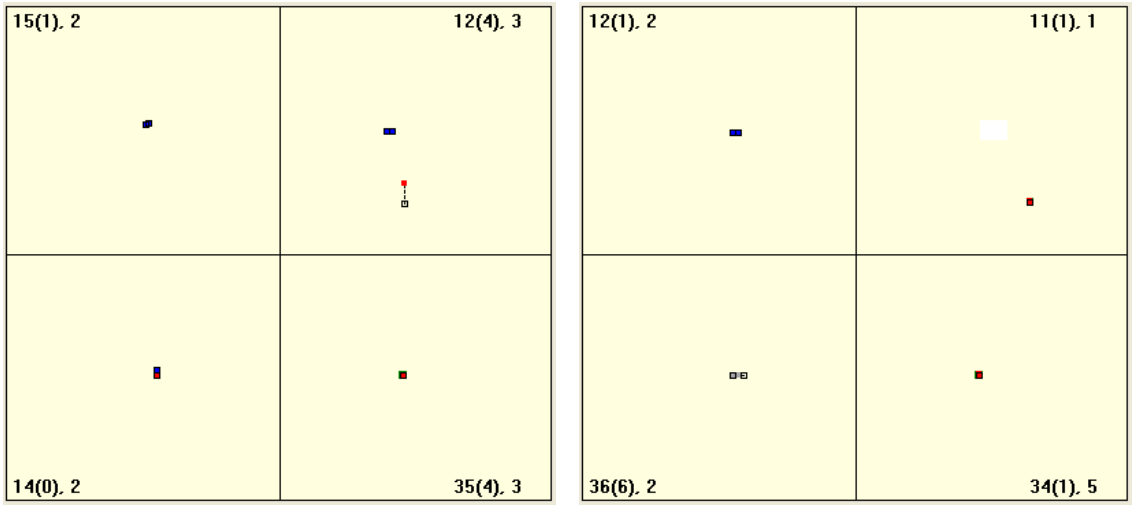


Fig. 9.33. Gradual forgetting of class number 1 in case when its trace in memory is not enforced systematically during consecutive self-learning process

Perhaps you realize that similar situation occurs when it comes to your own mind – for instance, on holiday you might take up botany and learn names of new plants, how to identify and classify them. But if you don't revise them and solidify your knowledge soon you will forget what you learned; the old information will be blurred by new knowledge (e.g. ability to identify new car brands) and as a result next summer you will say: *What a beautiful meadow! Fantastic flowers!* ***I used to know*** *the names of every single one of them…*

9.7. What kind of input data triggers a self-learning process?

(Translation by Weronika Łabaj, weronika.labaj@googlemail.com)

Let's leave aside the sad problem of forgetting and continue our investigation of how a self-learning neural network works. During experiments with the application **Example 10b** you could observe a disturbing phenomenon: because of the impulsiveness of the self-learning process the representations of some classes might be stronger (that is many neurons detect and recognize objects belonging to those classes), while others might have a very weak representation (or none at all!). This is a huge shortcoming of the self-learning methods discussed here and we will elaborate on it in the section 9.8.

Now I want to go back a little and show you that the processes of self-organization and self-learning occur only when some patterns exist in the input data sequence, patterns that the network can be based on. In the experiments with the Application Example 10b the situation was very neat and well organized because objects „shown" to the network belonged to a specific number (usually four) of well defined classes. Furthermore, those specific classes occupy distinct areas; namely the central parts of four quarters of the quarter system. The objects appeared in a random sequence but were not located in completely random places. Each time the point presented to the network was located (with some deviation) in the central area of the given quarter, so it was an example of a Martian-male, Marian-female etc. (no necessarily a completely typical example, since the typical one will be located exactly in the central point of the quarter). When using that kind of objects representation you could observe a familiar phenomenon; the self-learning process caused neurons to form groups, more and more specialized in the recognition of exactly those „patterns": the combination of many object representations from one class, slightly varied, variations of the same perfect pattern.

Let's consider now how behaves a network when points used during learning are located completely randomly. Let's go back to the example with the space probe sent to Mars. Let's say that Martians do not exist and that the lander receives from its sensors only pictures of random shapes created by a wind out of the martian dust. Thanks to the built-in feature in the Application Example 10b you can easily observe it: during learning process activate the RND learning option (check the checkbox since this option is disabled by default). From now on you will not see any more in which quarter the learning sample is located (Fig. 9.34) since it is not important (providing that you enable random points selection from all quarters – just as in Fig. 9.34) – the sample object will be located here and there, just anywhere in the coordinate system, without any order.

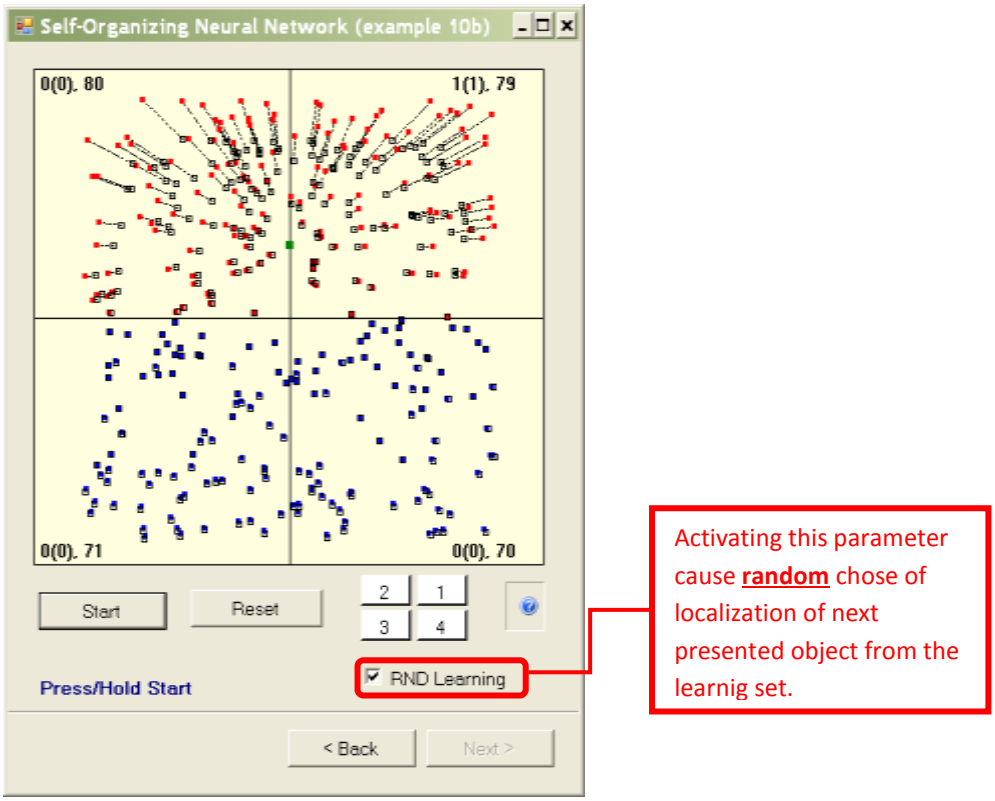


Fig. 9.34. The self-learning process in the beginning, random input objects

Because of the lack of order there will be no information in the input data sequence. One time neurons will be attracted to the centre of the area, the other time they will be pushed outside (Fig. 9.35).

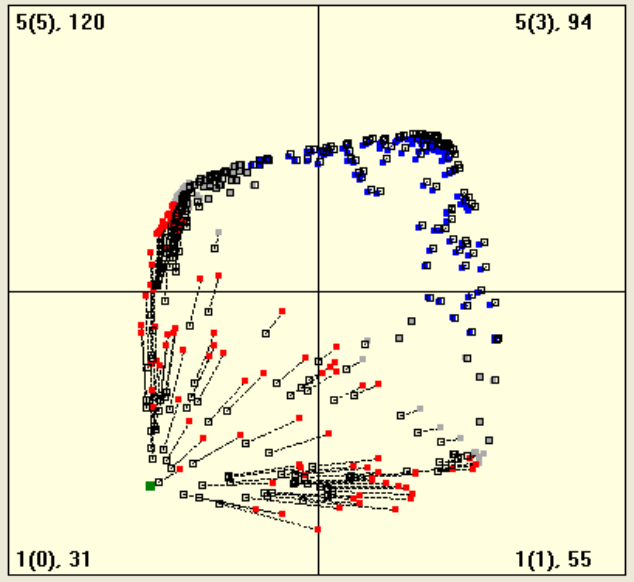


Fig. 9.35. Chaotic neuron movements when random objects are shown

In such a case the self-learning process will not lead to the creation of any visible groups of neurons, but neurons will rather form a huge circle, because inside of that circle the mean signal pattern will be located. When you watch the self-learning process in this special case for long enough you will

notice that the size and location of the circle eventually changes with every next presentation of the learning sample (Fig. 9.36), but never will any groups of neurons emerge – because there is not such a „grouping” in the input data too.

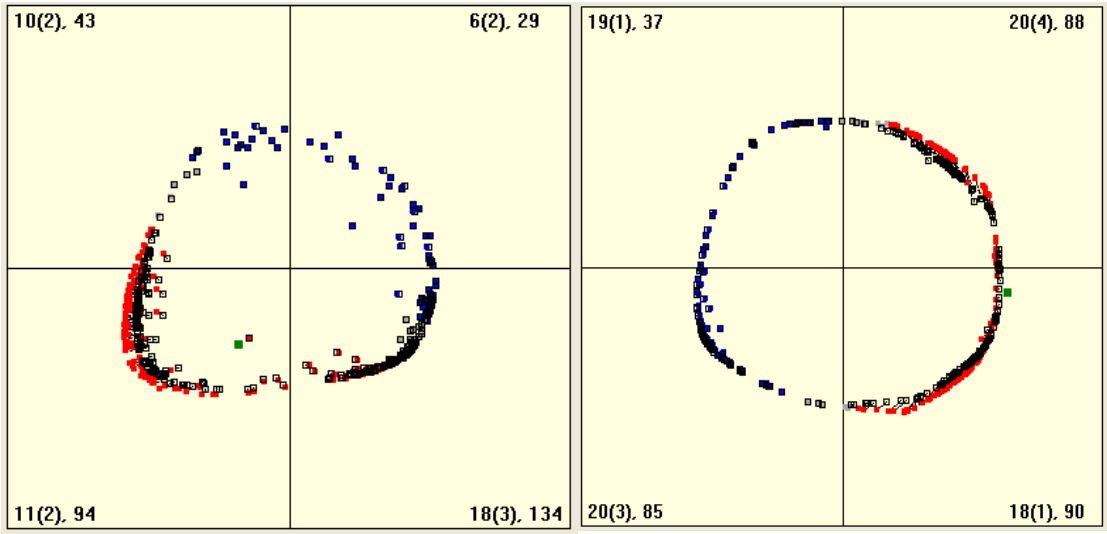


Fig. 9.36. Advanced stage of chaotic learning

The conclusion is both optimistic and pessimistic at the same time.

The optimistic part is that the self-learning process of a neural network may help in finding unknown patterns which are present in the input data and users do not have to know what and how many of those patterns are present. Sometimes it is also said that the self-learning network can answer unasked questions and speaking even more generally one can say that such networks not only accumulate knowledge (during learning), but also discover knowledge. This is awesome since in a computer science we have many good tools that are capable of providing good answers to good questions (such as Internet and database systems). Sometimes we can even obtain a good answer to the stupid question - that also happens to be useful. But there are not many tools for finding answers to unasked questions. The self-learning neural network has such capabilities and that is a great feature. Sometimes the process of finding answers to unasked questions is called Data Mining and is used for example in determining customers behaviours in the supermarket or determining preferences of mobile phone users. The marketing specialists can make use of even the smallest piece of data about repeatable and common customer behaviours, so there is a huge demand for that kind of information.
The pessimistic part is that if the network is self-learning with the data that does not contain any valuable information, for example if input values are completely random (devoid of any explicit or implicit meaning) then even the longest learning process will not provide any meaningful results.
Why do I consider this sad? Because that means that the neural network cannot replace us in coming up with new ideas that „come out of nowhere”. This is sad.
Or is it?

9.8. What do we gain from competition?

(Translation by Weronika Łabaj, weronika.labaj@googlemail.com)

Any type of a neural network can be self-learned, however most interesting results can be gained from enriching the self-learning process with competition. The competition between neurons should not be new for you. In section 4.8 I already described (do you remember? If not go back to the

application **Example 02**) how do networks in which neurons „compete" with each other look like and work. As you probably remember, in such competitive networks all neurons receive input signals (generally - the same signals, since such networks are usually one-layered), then all neurons calculate the sums of those signals (of course, they are multiplied by some weights that vary with each neuron). Then all values calculated by particular neurons are compared and the „winner" is found – that is the neuron which produced the strongest output value for the given input.

As you probably remember, the output value is the higher the better the accordance between the input signal and the internal pattern of the neuron. Therefore if you know the weights of the neurons you can predict which of them will win in the case of showing samples that lie in the particular areas of the input signal space. You could easily make that prediction, because only the neuron which internal knowledge is in accordance with the current input signal will win the competition and only its output signal will be sent to the output of the whole network. Outputs of all the other neurons will be ignored. Of course such a „success" of the neuron is short-lived, because the next moment new input data arrives and some other neuron „wins" the competition. There is nothing surprising there, because the map showing the arrangement of the weights values determines which neuron will be the winner for any given input signal – it will be the very same neuron which weight values vector is the most similar to the vector representing input signal.

There are a few consequences of winning the competition by one of the neurons. Firstly, in most networks of this type only one neuron has a non-zero output signal (usually its value is 1). The output signals of all other neurons are zeroed, what is known as WTA (Winner Takes All).

Furthermore, the self-learning process usually concerns only the „winner". Its (and only its!) weights values are altered in such a way that the next time the same input signal is presented the same „winning" neuron will produce even more „convincing" output (the output value will be higher).

Why is that?

To answer this question let's examine carefully what exactly happens in a self-learning network with a competition. On the input we have the object represented by its input signals. Those signals are propagated to all neurons and form a „combined stimulation". In the simplest case it is just a sum of input signal multiplied by the weights values, but we can apply the same rule for neurons with non-linear characteristics. The more weights values of the neuron are similar to the input signal the stronger the „combined stimulation" on the output of this neuron. We have already said that the sets of weights values can be treated as input signals „patterns" to which each neuron is particularly sensible. Therefore the more input signal is similar to the pattern stored in the neuron the stronger the output when this signal is used as an input. So when one of the neurons becomes the „winner" it means that its „internal pattern" is the most similar to the particular input signal out of all neurons.

But why is it similar?

In the beginning it might be the result of a random weights values initialization. In each network initial values of the weights values are random. Those randomly assigned values are more or less similar to the input signals used during learning process. Some neurons have – accidentally – an „innate bias" towards recognition of some objects and – also accidentally – an „aversion" towards others. Later on the learning process forces internal patterns to become more and more similar to some kinds of objects with each step of learning. The randomness disappears and neurons specialize in the recognition of particular classes of objects.

At this stage if a neuron „won" during recognition of a letter A it is even more probable that it will win once more when a letter A is presented on the input, even if it is slightly different from the

previous sample, for example written by another person. In the beginning we always start with randomness – neurons themselves decide which of them should recognize a letter A, which B, which should signal that particular character is not a letter but - for example – a fingerprint. The self-learning process always only reinforces and polishes the natural bias (again: randomly assigned during initial values generation).

This happens in every self-learning network so what is the meaning of the competition?

Thanks to the competition the self-learning process might be more effective and efficient.

Since initial values of weights are random then it might happen that a few neurons are „biased" towards the same class of objects. The normal process lacking the competition will be strengthening those „biases" simultaneously in all those neurons. Eventually there will be no variety between behaviors of various parts of the network (that is particular neurons), quite the contrary – the various parts will become more and more similar. You have seen exactly that phenomenon during experiments with the application **Example 10b**.

However, when we introduce a competition the situation changes completely. Each time there will be some neuron at least slightly more suitable for recognizing currently shown object than its „competitors". The natural consequence is that a neuron which weights values are (accidentally) most similar to the currently presented object will become the „winner". If this neuron (and only this one) will „be learning" in this particular step then its „inborn bias" will be - during learning process - further developed and strengthened, the „competition" will stay behind and will compete only for recognizing other classes of objects.

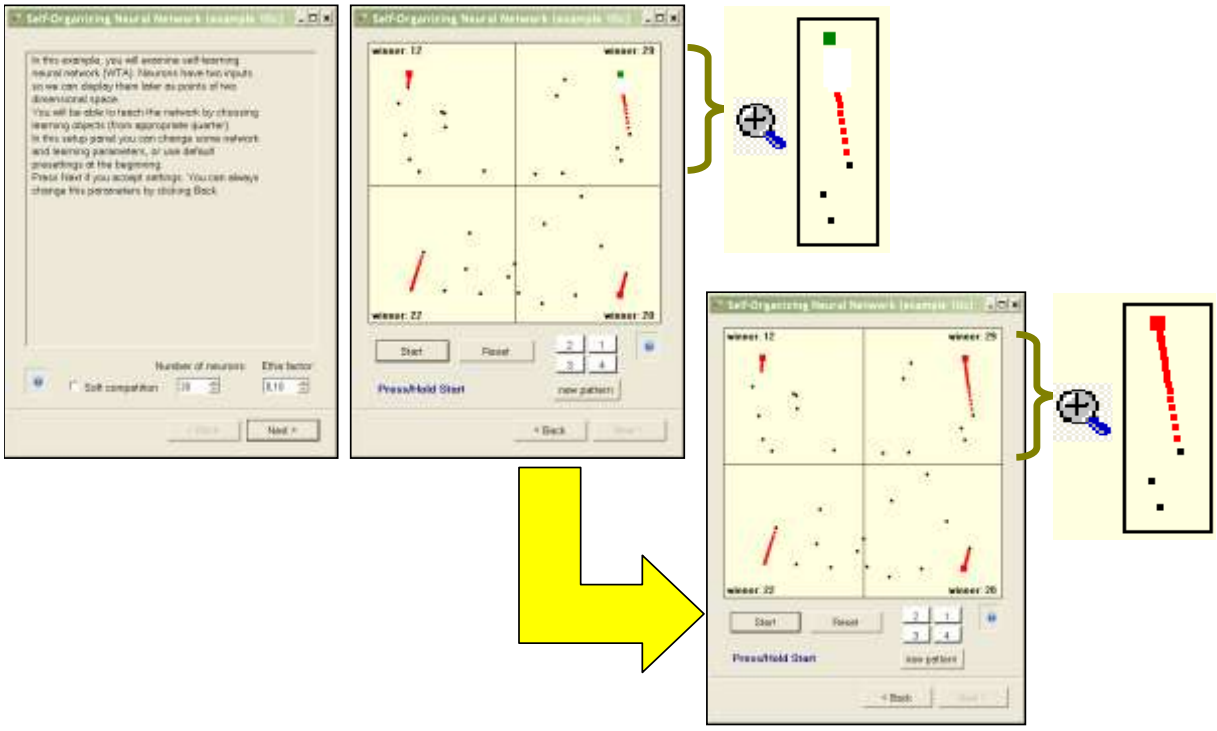


Fig. 9.37. Parameters window and self-learning process visualization in a network with competition – before and after eventual success. Application **Example 10c.**

You can observe the self-learning process with a competition using the Application Example 10c. In this application I used a competitive learning in the task similar to the previous ones from this

chapter (recognizing Marian males, females, kids and animals). However, this time the learning principle is different so you will observe completely new, different behaviors.

When started the application will display a window with parameters, where you can (among others) specify the number of neurons in the network. I recommend to manipulate only this parameter at first, because if you change many parameters it will be too easy to get lost.

The rules of determining number of neurons are simple: the less neurons (for example 30) the more spectacular the self-learning with a competition - you easily observe that all neurons but the winner „stay still" and only the winner is learning. I enabled tracing changes of location for learning neurons so you can observe their „trajectory". In the pictures generated by the Application **Example 10c** the neuron is presented as a big red square when it gets to its final location. You can consider it as a sign of completion of the learning process for the particular class (Fig. 9.37).

When the self-learning process starts you will see that only one neuron will be „attracted" to each point in which objects belonging to the particular class appear. This neuron will eventually become a perfect detector for objects that belong to this class (you will see a big red square in a place where objects of that class typically appeared during learning process). If you click Start button again you will activate a „step by step" feature – just like in a previous application (if you click Start button and hold it then the self-learning process will become automatic). Observing trajectories of moving weights values vectors of specific neurons (and reading messages shown in each quarter that inform which neuron „wins" in each step) you can notice that in each quarter one neuron is chosen that wins every time when samples from a given quarter are presented. Moreover only this neuron changes its location, moving towards a presented pattern. Eventually it reaches its final location and stops moving (Fig. 9.38).

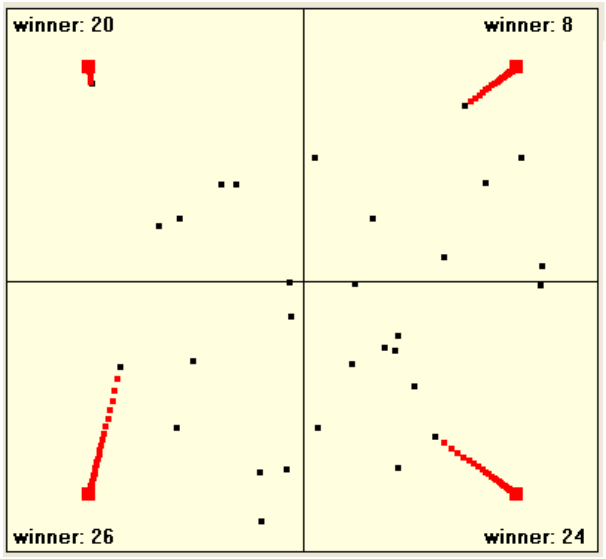


Fig. 9.38. Self-learning process in a network with competition

When the number of neurons is huge it is more difficult to observe the self-learning process with competition because it has (contrary to the classic self-learning in big networks) a very local character (Fig. 9.39). It stems from the fact that with many neurons the distance from the nearest neighbor to the winning one is very short and therefore the trajectory of the „winner" is hardly visible.

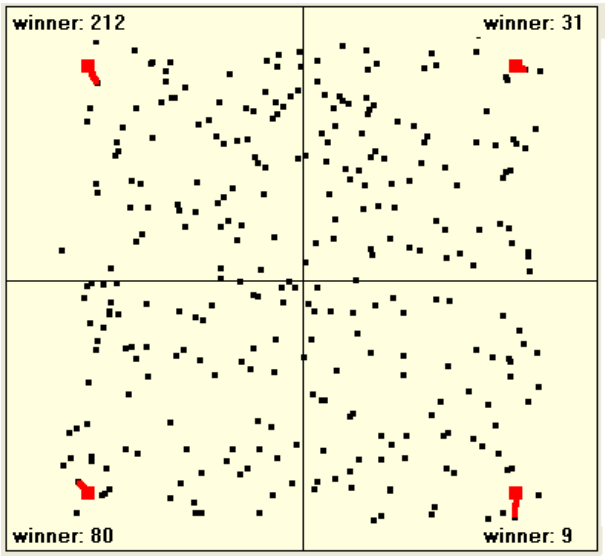


Fig. 9.39. Self-learning process with competition in a large network

On the other hand, with a small number of neurons (say 5) trajectories are spectacular and long, moreover, you can notice that thanks to the competition even very weak initial biases towards recognizing some classes of objects can be detected and strengthened during learning process – providing that „competitors" will have even weaker biases towards recognizing objects of a particular class (Fig. 9.40). Because of the visibility of the sequence of changing locations you will notice one more, quite interesting and general characteristic of the neural networks - that the learning is fastest and location changes are biggest in the beginning of the process.

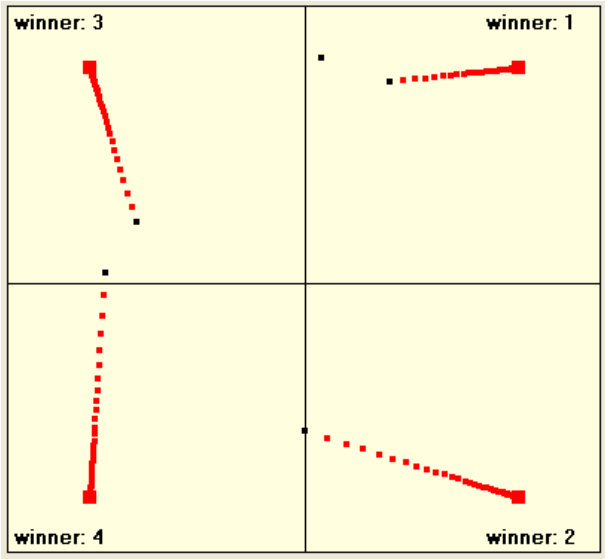


Fig. 9.40. Self-learning process with competition in a network with a low number of neurons

9.9. What kinds of self-learning we gain thanks to the competition?

(Translation by Weronika Łabaj, weronika.labaj@googlemail.com)

In the application **Example 10c** you can activate the parameter **Soft competition** (in the parameters window – Fig. 9.37, left) that enables „softer" competition. I recommend not using it from the very start (just leave the default setting). We will experiment with soft competition later, when you learn both the advantages and the disadvantages of the „hard" competition.

Thanks to the „hard" competition in the application **Example 10c** you could avoid cliques of neurons that had exactly the same preferences, just like those formed during experiments with the applications **Example 10a** and **Example 10b**. If you are lucky (especially in the case of working with networks that are composed of many more neurons than there are classes of recognized objects) then you could also avoid holes and „dead fields" in representations of input objects formed by particular neurons, in other words there were no objects that none of the neurons recognized. When competition is used then there is a high probability that in the network won't be any neuron recognizing multiple classes of objects and, at the same time, there won't be any class of objects not recognized by any neuron. Just like in an old proverb: *„Every Jack has his Jill*[8]*".*

When learning with a competition is applied then neurons other than the winner do not change their location, so they are ready for learning and accepting other patterns. Therefore, if objects from a completely new class suddenly appear during the learning process then there will **still** be some free neurons ready for learning objects from this new class and improving in identifying them. You can easily see that on your screen, because there is an option in the application **Example 10c** that allows you to demand a new class of objects to appear. To do that during simulation click the **new pattern** button. By clicking the **new pattern** button a few times you can lead your network to recognizing many more classes of objects than 4 types of Martians (do not ask me, however, what those other classes represent – my imagination has its limits!). Thanks to the learning with competition each of those many classes of input objects gets a private „guardian" - the neuron which from that moment on will identify with this class (Fig. 9.41). Usually there will be also some free neurons left, able to learn new patterns, if any appears in the future.

[8] You know that if not for the *WTA* competition (that stems from monogamy required in our country, that lacks any biological justification) then the intellectual cream of the crop will attract all the girls (that is computer scientists). Because of the competition some other guys can also be lucky...

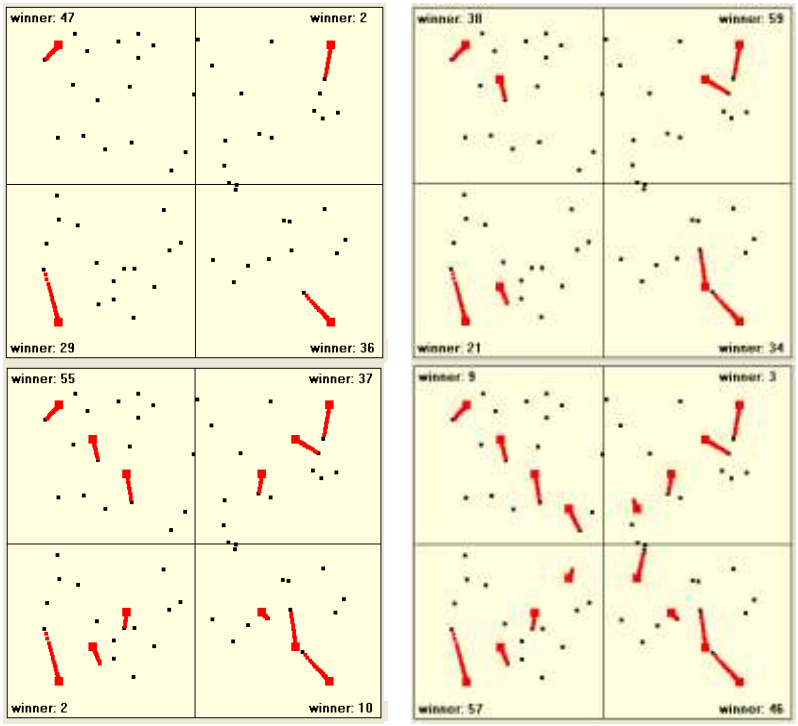


Fig. 9.41. The opportunity to teach recognition of multiple patterns in a network with competition

Unfortunately, the pure self-learning process combined with the „hard" competition may lead to some abnormalities. When you initiate the self-learning process with a low number of neurons then for each of the classes of objects (four in my sample application) there will be one neuron, that will identify with and recognize one class. If new classes of objects will be introduced later (for example when you click the **new pattern** button) then it might happen that the „winner" will be a neuron that already has specialized in recognizing some other class! That in turn might lead to the situation in which some class of input objects, that gained a strong representation among neurons suddenly loses it[9]! It is a well known situation from our daily lives – new information replaces the old one.

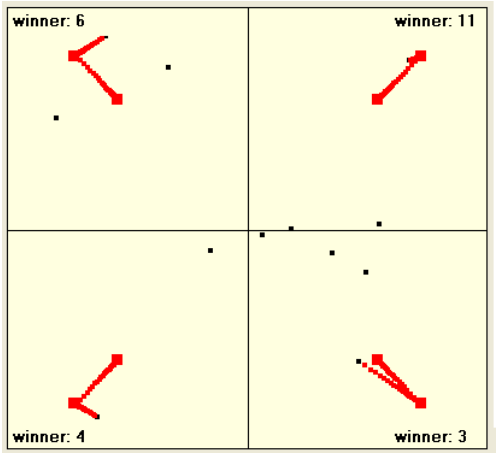


Fig. 9.42. Replacing previously learned skills by new information

[9] Continuing our matrimonial analogy it can be said that we are watching the birth of a betrayal and a divorce.

In the picture 9.42 is shown an example of such an effect, in this case it is very strong - **all** previously trained neurons recognizing some objects after being shown new objects „re-learned" and started recognizing those new objects. It is not a typical situation. Usually there are more neurons than the number of recognized classes of objects and the „kidnapping effect" is rather rare – it happens in one or two classes out of a dozen (Fig. 3.43). That explains why when using our own neural networks – that is our brains – only some, not so many details disappear from our memory, replaced by other, more intense memories. Unfortunately, usually things that „get lost" are the most important and they get lost right when most needed!

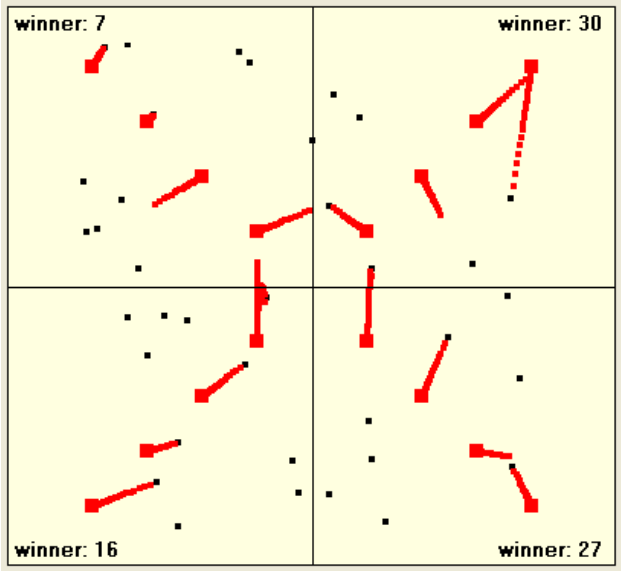


Fig. 9.43. In a network with „hard" competition very few previously learned patterns are lost

During experiments with the application **Example 10c** you will surely notice further „kidnapping-related" phenomenon. New objects might „kidnap" neurons that already were in a stable „relationship" with some class of objects; it might happen even if there are still many free neurons, if those free neurons are more distant. In the sample application the phenomenon gets more intense when firstly one class of objects is shown and only when neurons start identifying with this group then new classes of objects are shown. Then those new objects sometimes „steal" neurons from established classes. If all objects were shown at the same time then a competition will be present, showing in „attracting" neurons once to the first group then to the other (you can see that in the application **Example 10c** when you start the self-learning process with a very low number of neurons, see Fig. 9.44).

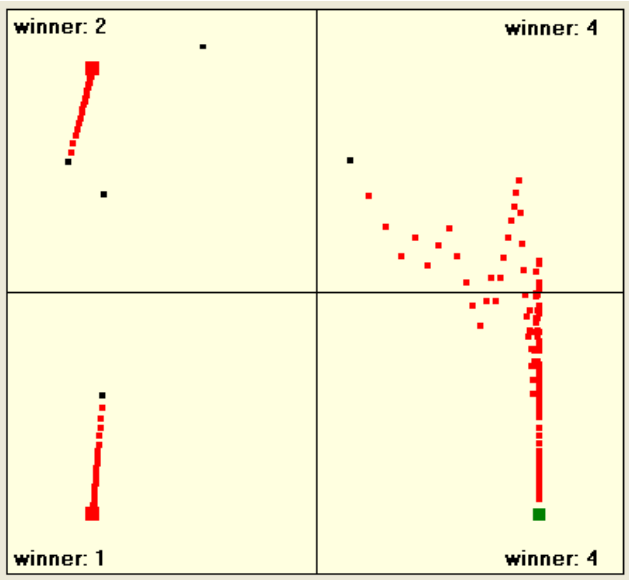


Fig. 9.44. „Attracting" a neuron between objects of classes 1. and 4.

In such a case it is possible that the network will learn **nothing**, even in a very long self-learning process. That also probably sounds familiar; think about school problems when a few exams were scheduled for one day - for example mathematics, physics, geography and history - you were probably not able to prepare well for **any** of those subjects!

The solution to that problem might be „softening" the competition. You can do this when you go back to the parameters window (click the **Back** button). The window is shown in the Figure 9.37, left. Find and activate the **Soft competition** parameter (check the checkbox). From this moment on the application will apply softer limited competition, that means that the winning neuron will be chosen only if the value of the output signal will be exceptionally high[10]. That means that neurons will be evenly divided between classes of recognized objects. Furthermore, there will be no kidnapping of neurons that already belong to some other class.

Unfortunately – instead of those you will see another worrying phenomenon: under some circumstances (especially in networks with a low number of neurons) there might be **no** winner among the neurons when a particular picture is shown[11] (Fig. 9.45).

[10] One can say that the softer competition is like partners that have to love each other a lot in order to get married. Weak and short fascination does not lead to any long-lived consequences (such as marriage).

[11] There is an analogy also for this situation: Surely you know old maids and bachelors. Their existence is exactly the effect of combining the effect of the **competition** (monogamy) with its **soft version** (a strong affection is needed to get married).

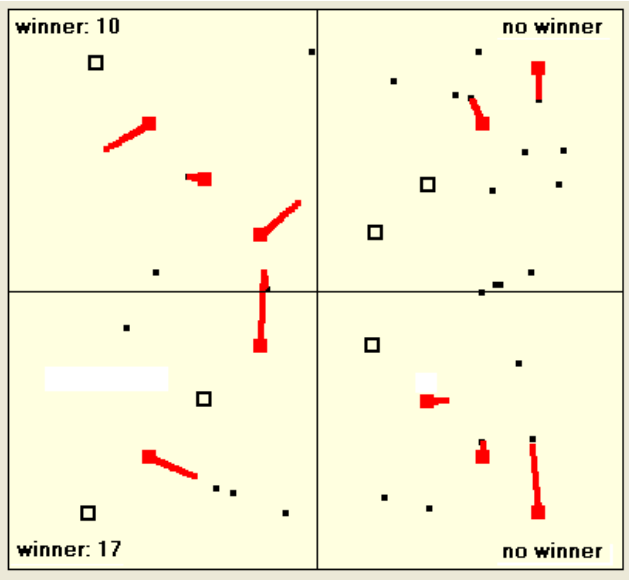


Fig. 9.45. Omission of some classes in a network with softened competition

Use the application **Example 10c** with a soft competition to examine and analyze this phenomenon carefully. It will be easy, because with this option the application uses special markers that show the location of patterns of classes that are omitted (that is classes for which there is no winner among neurons).

You will discover that when there is no strong competition then such omissions happen very often, especially in networks with a small number of neurons. In such cases for most of classes in the network – completely voluntarily! - patterns will be created that allow later for an automatic recognition of signals that belong to the specific classes, but for the „unlucky" class that no neuron want to detect - there will be **no** specialized detectors.

- And what?

- Nothing!

Have you ever seen a person that has a flair for history, geography and languages but mathematics is beyond their abilities?

9.10. Questions and exercises

(Translation by Weronika Łabaj, weronika.labaj@googlemail.com)

1. Could you explain for whom, when and what for the self-learning neural network might be useful?

2. What determines whether points representing neurons in the application **Example 10a** will be blue or red?

3. Taking into consideration experiments described in this book and your own observations try to formulate your own hypothesis on the influence that the inborn talents (represented by the initial, randomly organized weights vectors) have for the gained knowledge (represented by the self-learning process). Do you think that systems having a small number of neurons

(such as primitive animals) are strongly determined by their instincts and inborn qualities, while more complex systems with many neurons (such as a human brain) are much stronger influenced by the external factors related to the personal experiences and education? Try to find arguments for both points of view using self-learning networks of various sizes.

4. Describe the features of self-learning networks with a very low and a very high value of learning factor ***Etha***. Do we observe similar phenomenon in animals brains? When answering this question take into account that learning is easier for younger brains and it becomes more and more difficult with age (that process is represented by the diminishing values of ***Etha***), for example a young puppy has a brain that can be modeled with a network with a very high value of ***Etha*** and is different from the brain of an old dog (in which case ***Etha*** might even be close to zero). Try to think about both good and bad sides of this fact.

5. Do you think that a self-learning network that has „imagination" capabilities described in this chapter can come up with a completely new term or idea, that has no representation in a real world or - more precisely - in a space of input signals that the network uses during self-learning process?

6. Can you imagine a real-life situation that represents the experience described in the section 9.7? Try to think about such a situation and consider what influence it would have for an animal or human brain (especially young kid).

7. Do you think that the **forgetting** mechanism caused by „over-learning" described in the section 9.6 explains all the issues related with this process? And apart from that – is the forgetting of information that is not constantly updated good from a biological point of view or not?

8. Which adverse phenomenon that occur during self-learning of a neural network can be eliminated by introducing competition and which will remain?

9. In networks with competition the WTA rule (*Winner Takes All*) is sometimes replaced by WTM rule (*Winner Takes Most*). What does it mean? What do you think? Check on the Internet what it is.

10. Let's say that you want to design a self-learning network that is able to create patterns for 8 classes. Prepare an experiment to help you determine how many more neurons (more than the minimum - 8) should such a network have at the beginning of the self-learning, so no class will be omitted and at the same time the number of „free" neurons will be as low as possible.

11. **Advanced** Modify the Application **Example 10b** to enable defining how the ***Etha*** parameter will change over time. Allow also for planning appearance of some objects only in some period during the „lifetime" of a network (for example only during „childhood" or only later, when a network is already trained and experienced). Run a series of experiments with such a „real-life" application, then write down your observations and try to come up with conclusions. What observed phenomenon have their representations in a real-life and what are only the result of simplifying real neural networks (brains)?

12. **For advanced readers:** Modify the application **Example 10c** to enable competition with a „conscience". „Conscience" means that the neuron that wins very often gradually gives up and lets other neurons win. Compare such a behavior with the „clean, hard competition". What conclusions can you derive? Do those conclusions apply only to the neural networks or could they also be related to some real-life processes (for example in economics)?

10. Self-organizing neural networks

10.1. What is the structure of the neural network, at which you will create mappings, being result of self-organizing?

(Translated by: Michał Głamowski, glamowskim@gmail.com)

Neural network can be taught with a teacher or can learn by itself. You already know that, you did it. However, now you are going to get to know networks, which not only learn by themselves (without a teacher), but in addition, agree (on the basis of the interaction) functioning of all neurons, that their cumulative resultant acting brings some new quality. It is about automatically generated by the network complex mapping from a set of input signals into set of output signals. This mapping generally cannot be predicted or somehow determined so the networks discussed here are much more self-reliant and independent than the networks, with which you had to do before, because their properties are only slightly imposed by the creator of the network, while the essence of their operation and the final mapping, being result of their work - is mostly determined by mentioned process of self-organization. Details of the process will be illustrated in a minute by the next program. Before that though, I'd like to briefly tell you, what all that can be used to, so that you won't limit your interest only to watching images, but also try to imagine self-organizing networks as serious tools for serious purposes.

The mapping is a quite rich and complex mathematical concept, which has - depending on circumstances - several specific characteristics and properties. I don't want to bore you with information on the theory, so I will try to briefly (but unfortunately, not very strictly) discuss the point, without going into important, but tough and complicated details. I do not know if I succeed, so let's agree - if you find it too hard (or too boring...) during reading these explanations - you can leave without regret all this theory and start reading specific and detailed information on how to build a self-organizing network by yourself and how to examine its properties, in the next subsection. However, if you decide to make an effort of reading this theoretical introduction - you will learn what such self-organizing networks are built for. You can live without it, but you'll get more satisfaction later reading and watching what such a network is doing, knowing also - what it might be useful for.

I assume that if you read this, you are eager adventurer, and I can introduce you into more difficult things. Remember - you wanted this!

Let's start by saying that in general we very often need to convert the input signals in the output signals according to certain rules. For example, building robots, to fulfill certain tasks, we must ensure that their control systems will be able to correctly convert the signals received by the sensors (video cameras, microphones, touch-replacement contact switches, ultrasonic sensors, approximation sensors, etc.) to signals controlling driving mechanisms of legs, graspers, arms, etc. (Fig. 10.1)

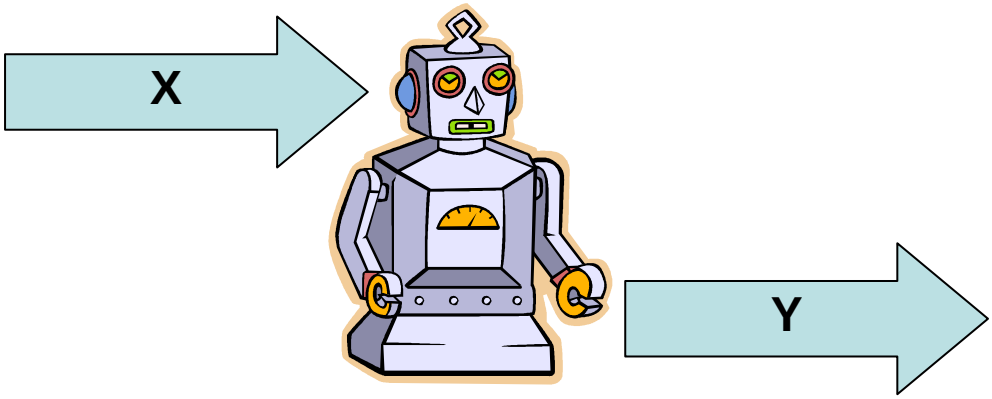


Fig. 10.1. Functioning of the considered robot is based on the mapping signals X to signals Y

Such conversion of any input signals into correct output signals is exactly what is called mapping.

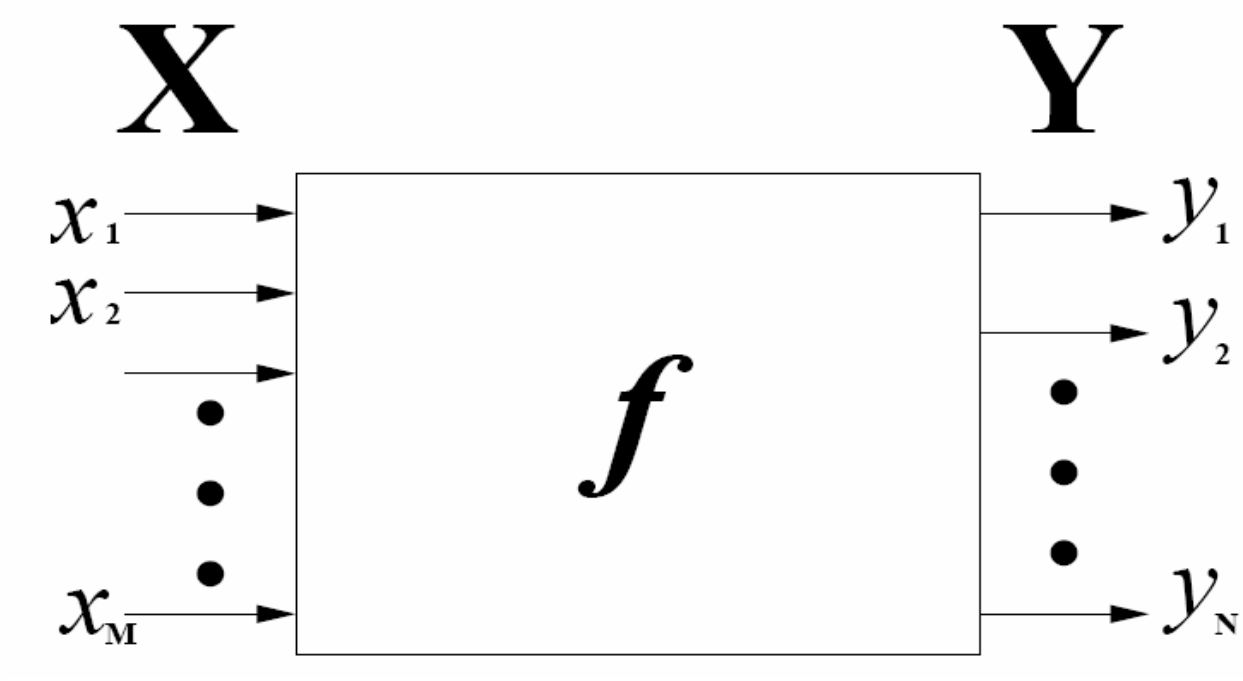


Fig. 10.2 General representation of the mapping

It of course may not be some random mapping, because it would be completely useless. In order to robot move correctly, perform meaningful tasks, respond properly to commands - the mapping, that binds stimuli recorded by sensors with concrete moves performed with the actuators, must be properly selected, programmed in detail, carefully thought out. Design engineer of robot (likewise design engineers of many other automatic systems) is hard put to determine a desired mapping, which is in fact essential content of a "brain" (controller) of the robot.

If there's only one input signal and only one output signal - this task is quite simple, and the mapping you need is a function that you undoubtedly know from school. However, if there are many sensors that are senses of robot (and in such cases there is always a lot of them!) and many actuators (and there must be many of them, if a robot is about to do something interesting) - the task becomes very hard, troublesome and time consuming. While you could "manually" set and describe all necessary mappings, in large and complex tasks, it might take the rest of your life.

This is the case that self-organizing neural networks may prove helpful. An example of such a network structure is shown in Figure 10.3.

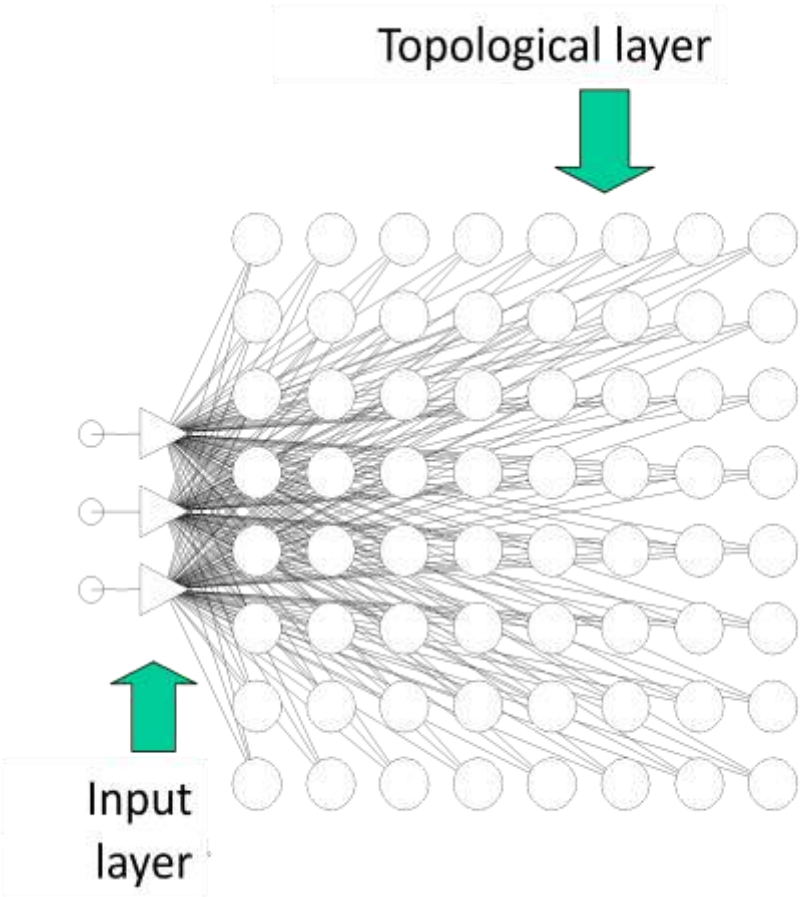


Fig. 10.3. Structure of the self-organizing neural network

Self-organizing networks have usually quite a lot of inputs. Figure 10.3 shows in fact only three inputs, but this is in order to not complicate the picture - the tangle of connections at thirty inputs would be impossible to trace! This type of networks used in practice have typically over a dozen (and often tens) of inputs, and their usefulness in the context of various applications is usually higher, the more input signals, a network can handle. But what is more important and more difficult: these networks tend to have more outputs, as they consist of vast numbers of neurons, forming together the so-called topological layer. Here the term "vast numbers of neurons" means at least tens and

often hundreds, sometimes even thousands of neurons - so this is really much more than in all the networks you have seen so far. How to understand what the network actually represents in this vast topological layer as a result of its work?

The concept that has already occurred many times in this book, namely the idea of rivalry and identification of the winner neurons, comes to aid here. After providing any specific input signal to the network, all the topological layer neurons calculate their output signals, which are their responses to this signal. Among these signals one is usually the largest - and the neuron, which produced this largest signal becomes the winner (Figure 10.4).

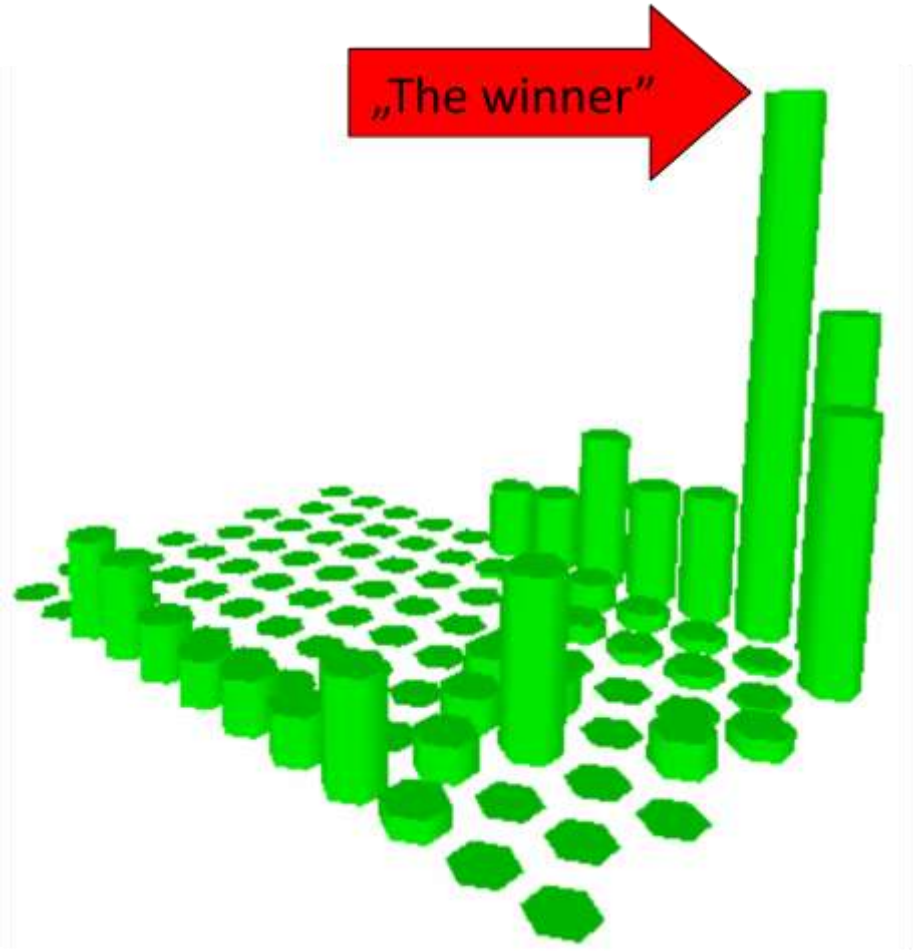


Fig. 10.4. Exemplary distribution of output signal values presented on topological layer of the network with selected signal (and neuron) called the winner

The process of self-organization, which will be thoroughly described in the following subsections, results, among others, that there is one and only one winning neuron and causes the predominance of "winner" over the "competitors" is very large and clear. Only if the network (after the process of self-organization) gets completely abnormal input signal (unlike any signal, which appeared in the teaching set) - it becomes difficult to determine the winner, because all the topological layer neurons produce very weak (and in fact little varied) signals. However, this is an unusual situation.

10.2. What is the self-organization in the network and what it might be helpful for?

(translation by Bartosz Wilczyński; b.wilczynski.89@gmail.com)

Even this small piece of knowledge you have gained in the previous section, allows you to find out what self-organizing networks are able to. They form - totally by their own, solely on the basis of observations of input data - certain representation of the set of input signals into a set of output signals, whereby the curve meet certain general criteria (I will discuss it in a moment), but in no way determined in advance by the creator of the network or by its user. Structure and properties of projection which we need should arise spontaneously in a coordinated self-learning process of all the elements of the network. Such spontaneous creation of needed signals projection is called self-organization. From a certain point of view, it is just another form of self-learning, but if you look at its effects (and you will see many of these, because the program used in this chapter gives you all the interesting pictures), you will probably admit that we are dealing with a clearly higher level of adaptation of the network, involving not only optimization of the parameters of each neuron separately, but - and this is just a novelty - the coordination of activities of neurons, giving to calculations highly desirable effects of **grouping** and **collectivity**.

Let us explain these concepts. The effect of grouping lies in the fact that the network in the process of self-organization is trying to divide the input data, distinguishing among them certain **classes of similarity**. This works so, that among the input objects (described by the input signals) such groups of them are detected (completely automatically!), where you can place the signals which are similar to each other in their group and which are clearly distinct from the signals assigned to other groups.

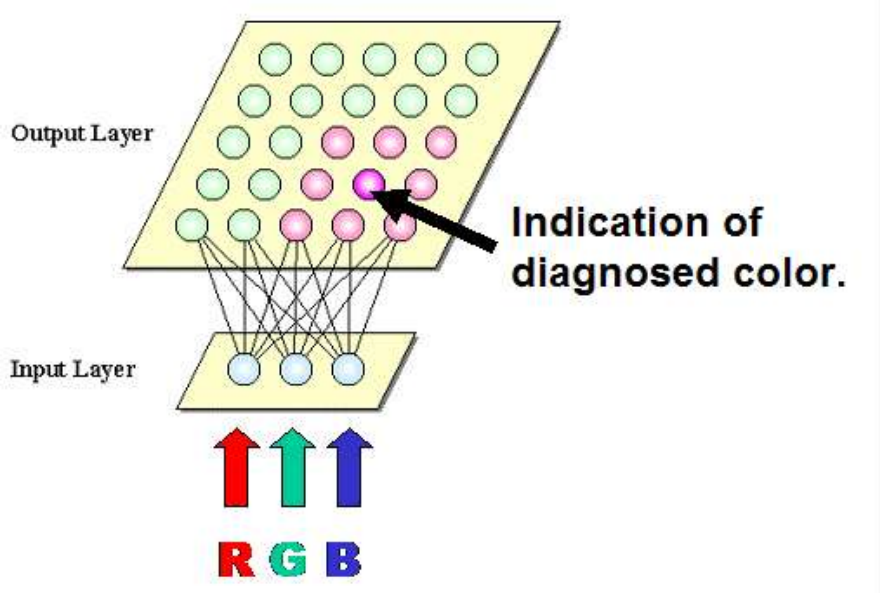


Figure 10.5 Network grouping the input signals (components corresponding to the typical coding of digital image) in the collections corresponding to different possible colors.

Such clustering of data is very useful in many applications, so a number of specialized mathematical techniques allowing the analysis and the creation of such groups of them (mostly statistical) were developed. Mentioned techniques are known under the English name of the *cluster analysis*. They are quite willingly used in the economy - for example, to detect which companies are similar to each

other and can therefore pose a similar return prognosis of investment, either in medicine - for the study which symptoms are indicatives of the different varieties of the same disease, and which already indicate the presence of new - perhaps not known - disease.

In Figure 10.5 you have an example of neural network which is able to group the input signals (color image pixels coded in the usual manner using the three components of RGB) according to the criteria of similarity of their colors. Figure 10.6 shows the effect of the network without self-organization process (on the left) and after the process of self-organization (on the right). Figure 10.6 was created in such a way that in every place where there is a topological layer neuron, was drawn a box filled with such color of the pixel input, which makes this particular neuron able to become a "winner".

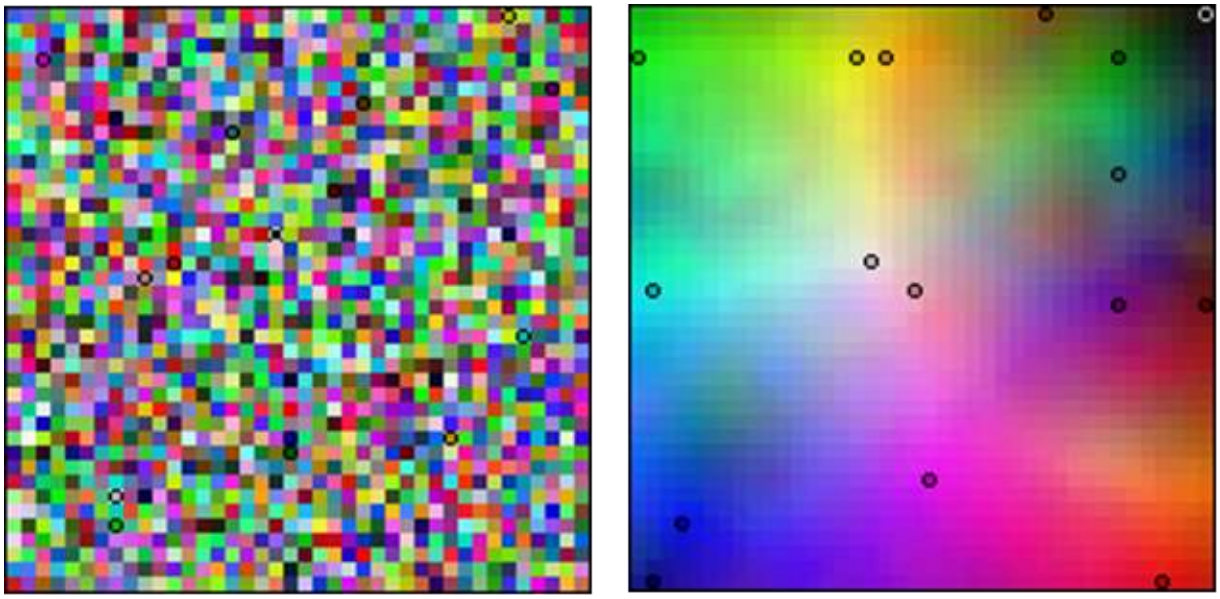


Figure 10.6. Activity projection of the network from Figure 10.5. On the left - before the process of self-organization, right - after the process of self-organization.

Grouping of image pixels together, having a similar color, is not a task that could give any special benefits, so you can see the figure 10.6 as a curiosity, and then you can shrug and say, "Okay, but what do we need that for?" Well, look in Figure 10.7. This figure is showing what was in the topological layer of the self-organizing network, which was told to compare data of different companies. At the entrance of the network various data was given: type of business, capital owned, number of employees, balance data sheet for several recent quarters, reflecting the profits gained or losses of the company etc. The network grouped these companies, and so placed the parameters that each neuron had a habit to become the "winner" when the information about "his" company was shown. Then - after waiting a year - those companies were examined. It turned out that some of them fell down or they are threatened with bankruptcy - those whose neurons are indicated on Figure 10.7 in red. Others on the contrary, proved to be affected by economic stagnation. Neurons, which are assigned to them in the automatic process of self-organization are indicated in Figure 10.7 with the blue color. And finally was a group of companies that have developed in harmony - you guess they are marked on green.

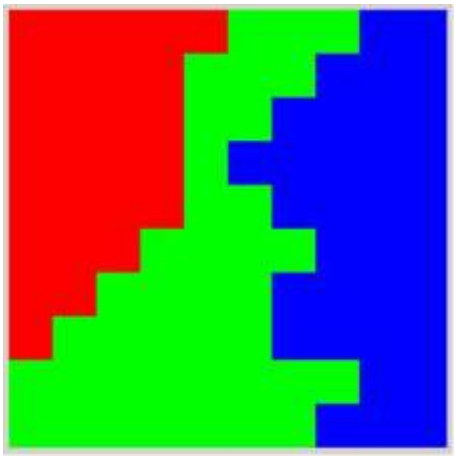


Figure 10.7. The effect of clustering in the self-organizing neural network. The description in the text.

Such a very impressive examples of self-organization, leading to the ordering of certain data, you can find plenty on the Internet - just type in Google the term "self-organizing networks" or better - a commonly used acronym SOM (Self-Organizing Maps). By doing this very quickly you can find out that tasks of grouping the input data described here into classes of similarity are needed and finds practical applications.

Neural networks engaged in the process of self-organization are therefore a very attractive tool to perform the task of grouping the input data and the creation of classes of similarity among them. The attractiveness of neural approach to the concerned problem is mainly that the self-organizing process can be carried out here absolutely automatically and spontaneously, and the creator of the network does not need to give any clues to it because the set of the necessary information is contained in the same input data (from which the network itself will extract the observation that some are similar to each other, while others do not). In addition, after the network is learnt how to cluster the input data, very useful systems arise - "winning" neurons, which specialize in identifying specific classes of input signals, so they are detectors and can be used as their indicators. Having talked briefly about what is the effect of clustering in self-organizing neural networks I will tell you even in a few words what I have meant by the **collectivity** of activity of the network.

Well, the networks in which the self-organization takes place are so organized that what a neuron identifies, to a large extent depends also on what other neurons (located in its vicinity) identify. In this way, **community** of neurons (or their collective) could process information in a fuller and richer way than each of the individual neurons taken. See Figure 10.8.

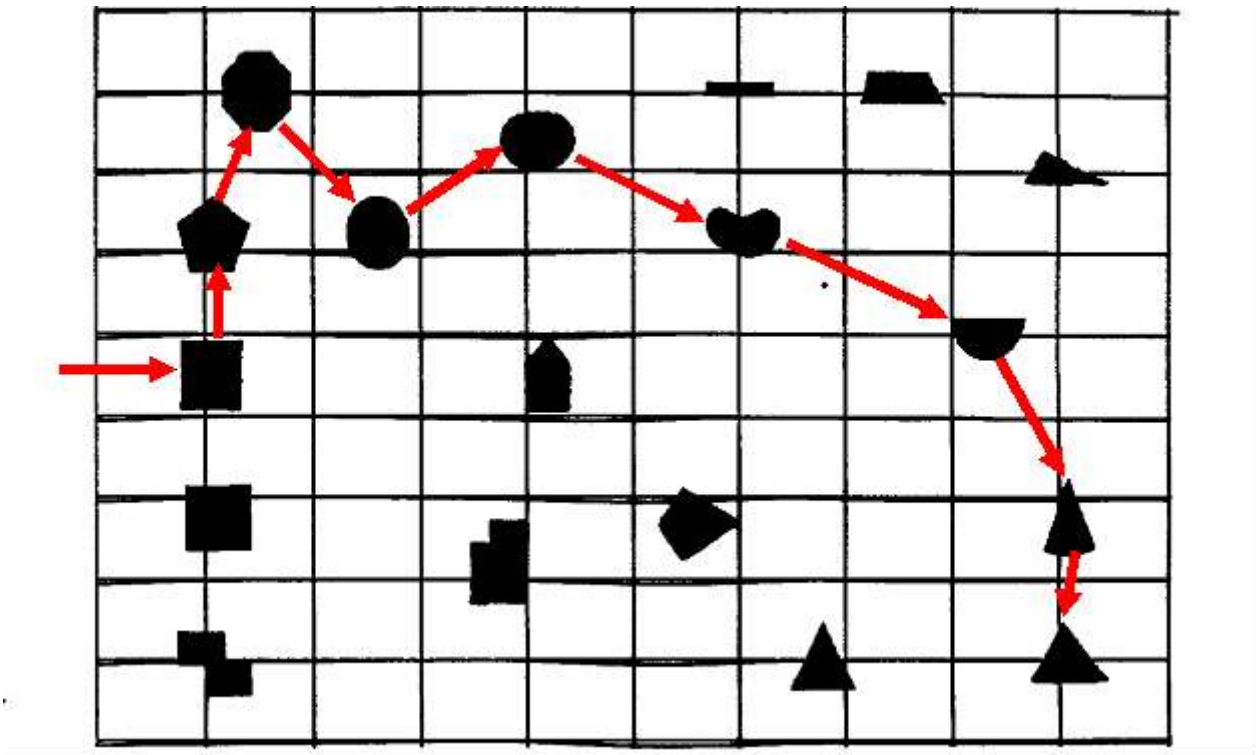


Figure 10.8. The effect of recognition of geometric figures of self-organizing neural network. The description in the text.

The figure shows the results of my research related to the recognition by the self-organizing neural network of simple geometrical figures, shown to it as the images. In places where the network placed (totally by its own!) the neurons indicating the different shapes of geometrical figures shown to it - there are pictures of the relevant figures. Make with me a brief tour of this map, and you'll find that this distribution of neurons is certainly not accidental!

We start - with red arrows, which I added to the figure - from the neuron, which becomes the "winner" whenever **a square** is shown to the network. Not far from it there is the neuron indicating **a pentagon**, and a little further - neuron signaling **an octagon**. Rightly: a pentagon is more similar to a square than an octagon! Near the octagon the neuron recognizing **a circle** was placed. From it short path leads to neuron signaling **an ellipse**, which in turn is not far from the neuron which understands **a semicircle**. Semicircle has a clear affinity for a smaller piece of the circle, called by us **a sector**, and this in turn is clearly similar to the shape of **a triangle**...

As you can see, the effect of collective self-organized networks is very interesting. What is more, this is the effect, which is a contribution to the broader considerations: Well system, or just the community of appropriately linked and cooperative elements makes it possible to obtain new forms of behavior and new forms of actions, much richer than might be expected taking into account each of the elements separately. For example, every single insect is quite stupid and primitive animal, and yet the community of insects (bee colony, nest, termite mound, etc.) are capable of deliberate, complex and undoubtedly intelligent actions. Once I describe my experience that I myself have collected during the computer modeling of bee colony and then I will write about it more - but not in this book.

Returning to the networks with self-organization, it is clear that they are quite convenient, useful and frequently used tool, and also - perhaps primarily - are very interesting object of research. You will learn details of the self-organization process ongoing in neural networks by studying precisely the material presented in this chapter and - as usual - doing the experiments with the program especially prepared for this purpose. The key to understanding and using of self-organized networks is the concept of **neighborhood** of neurons - and right now we will consider it for a moment.

10.3. How to implement a neighborhood in a network?

(translation Rafał Opiał, oorafal@gmail.com)

To truly understand the structure (and function!) of self-organizing network, first we try to compare its functioning, outlined above, with what a simple self-learning networks were able to do, as described in the previous chapter. Well, the result of the overwhelming influence of random initial values on the course of a simple self-learning process, described in Chapter 9, is that the creator of the ordinary self-learning network has no influence on which neurons what will be learned. Sometimes the distribution of neurons (after completing the self-learning) signaling certain events or certain phenomena can be very uncomfortable and adverse, but you cannot change it without "manual" intervention in the network behavior, which in the general case is quite difficult to do, and is also contradictory to adopted principle of self-learning. In turn by the introduction into network of **competition** alone, it may happen that only some (usually quite a few) neurons with large „innate abilities“ will be subject to the teaching process, while others remain outside the process and will be - from the viewpoint of the purpose for which network was organized - in fact lost. You could observe this, for example, in program: Example 10c, which outputted - among other things - information about which neurons (specific numbers) become „winners“ for each class and later on become detectors of class objects. These numbers appear strongly in a chaotic manner, without any relation to the location of detected objects. Whereas, the assignation of specific object detection function to a specific neuron, just that, and not to another - could be an important factor for increasing or decreasing the utility of a network. If - for example - some of the input signals are somehow similar to each other (in any sense), You may care, so that their appearance was signaled by a neighboring neurons in the network. In this way worked the self-organizing network, which results of work I showed you in Figure 10.8. Yet with „pure“ self-learning networks you do not have any influence for that and have to humbly accept what will happen.

A solution to all these worries is to introduce one more mechanism to self-learning networks: the neighborhood. For the first time the neighborhood of neurons in the networks applied (in the 70's) Finnish researcher Teuvo Kohonen, and therefore in the literature, networks which make use of the neurons neighborhood (and usually also involving competition between neurons) are called Kohonen Neural Networks.

I'll show you now, what this neighborhood is, and then tell you what it gives you.

So far, you have considered neurons in network as units largely independent of each other. Although they were linked together and transferred signals to each other (just like in a network) - but their relative **position** in the layers did not matter at all. At the most, for the ordering purposes there were introduced (used also for organizing calculations in programs simulating nets) the numbering of neurons - and that's it. Yet, in the self-organizing network which is under consideration here, it is very

important which neurons in the topological layer you consider **adjacent**, as this will significantly influence a behavior of the entire network.

We, most likely, associate neurons of a topological layer with points of some map, drawn or displayed on a computer screen, so typically we consider a **two-dimensional neighborhood**: neurons are considered as if they were placed in nodes of a regular grid composed of rows and columns. In such a grid, each neuron has at least four neighbors, two horizontal (left and right) and two vertical (top and bottom) - Figure 10.9.

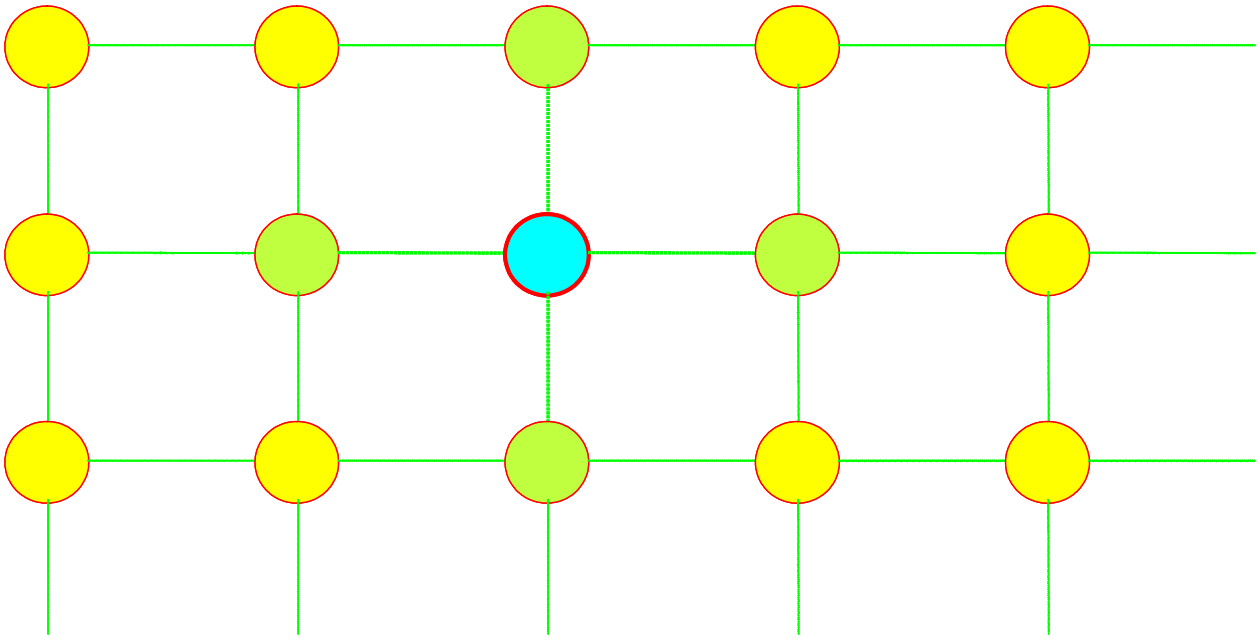


Fig.10.9. Simplest case of neuron neighborhood

If you need to - the neighborhood can be considered more broadly: also allow the neighbors on the diagonal (Fig. 10.10), or join into the neighborhood neurons located in further rows or columns (Figure 10.11).

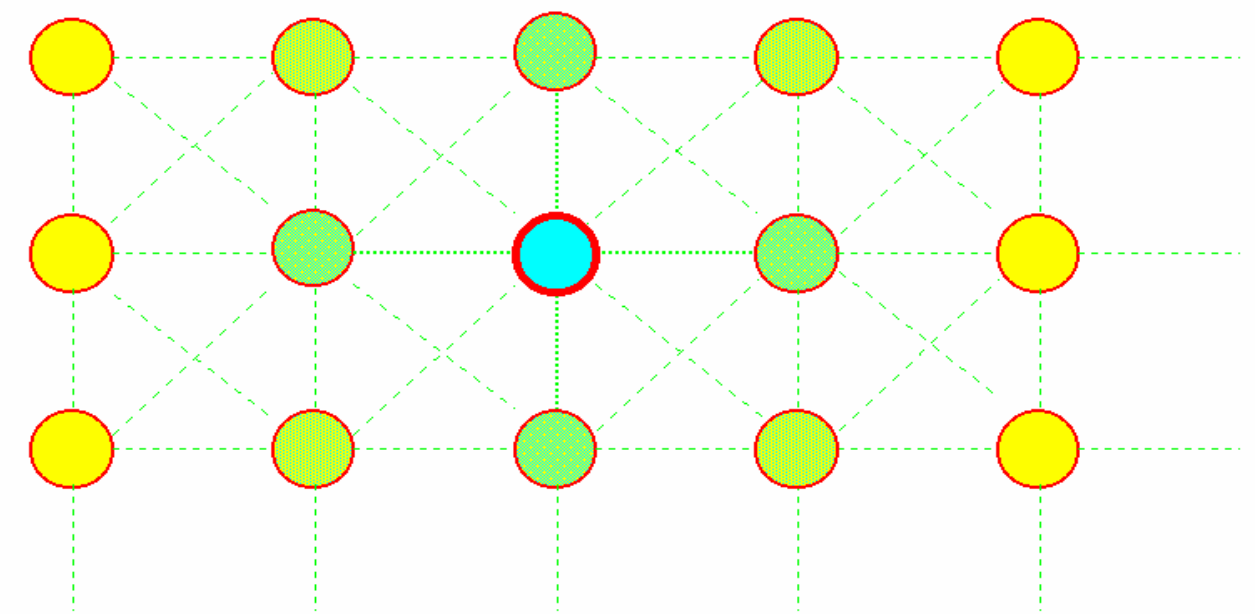


Fig. 10.10. More “rich” neighborhood

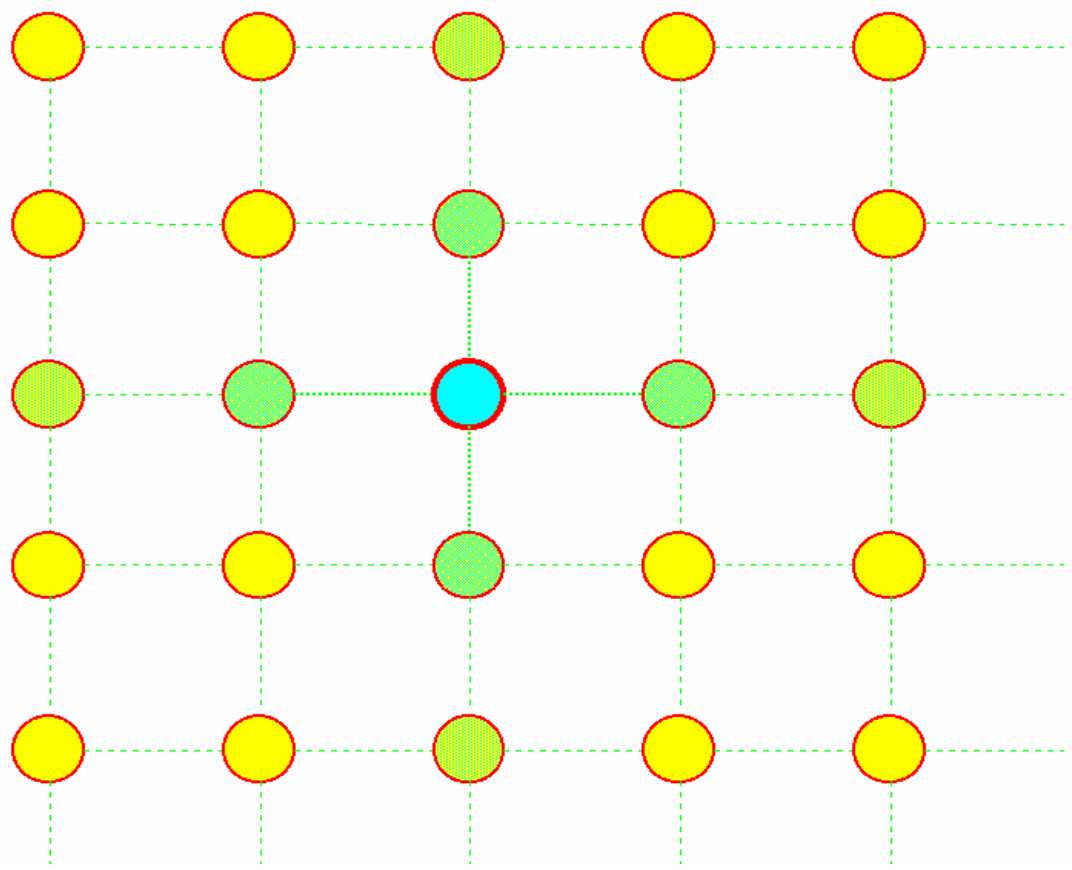


Fig. 10.11. Neighborhood with longer distance.

The way you will define a neighborhood is entirely up to you - for example you can describe a network, so that a neighborhood will be also one-dimensional neurons will then form a long chain, and each neuron will have the neighbors preceding it in the chain and neighbors who follow him - fig 10.12.

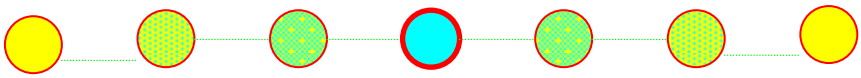


Fig. 10.12. One-dimensional neighborhood

In specialized applications of self-organizing networks can be applied a three-dimensional neighborhood (neurons with their neighbors look then just as atoms located in a crystal lattice - you have certainly seen some of these pictures already, and I do not want to exert myself and draw here anything like it, so imagine it yourself), it happen also to consider a neighborhood four-, five- or more dimensional (guess why I will not draw you this?). But definitely the most practical applications have networks one- and two- dimensional, this is why we will restrict all the further considerations only to them.

The neighborhood concerns, of course, all the neurons in a network: each neuron has a set of neighbors, and he in turn is a neighbor to other neurons. Only neurons located at the edge of the network do not have a full set of neighbors, but sometimes this can be remedied by making a special agreement (e.g. "closing" the network in such a way that the neurons of the upper edge are treated as neighbors to neurons from the lower edge; similarly can be closed left and right edges of the network).

10.4. What follows from the fact that some neurons we consider neighbor?

(translation Rafał Opiał, oorafal@gmail.com)

A fact that some neurons are considered to be adjacent (are neighbors) has a very important meaning. When, during a teaching process, certain neuron becomes a winner, and is subject to the teaching process – it's neighbors are being learnt along with it. Soon I will show you how it happens, but before, I'll remind you of how proceeds the teaching of a single neuron in self- learning networks (fig. 10.13).

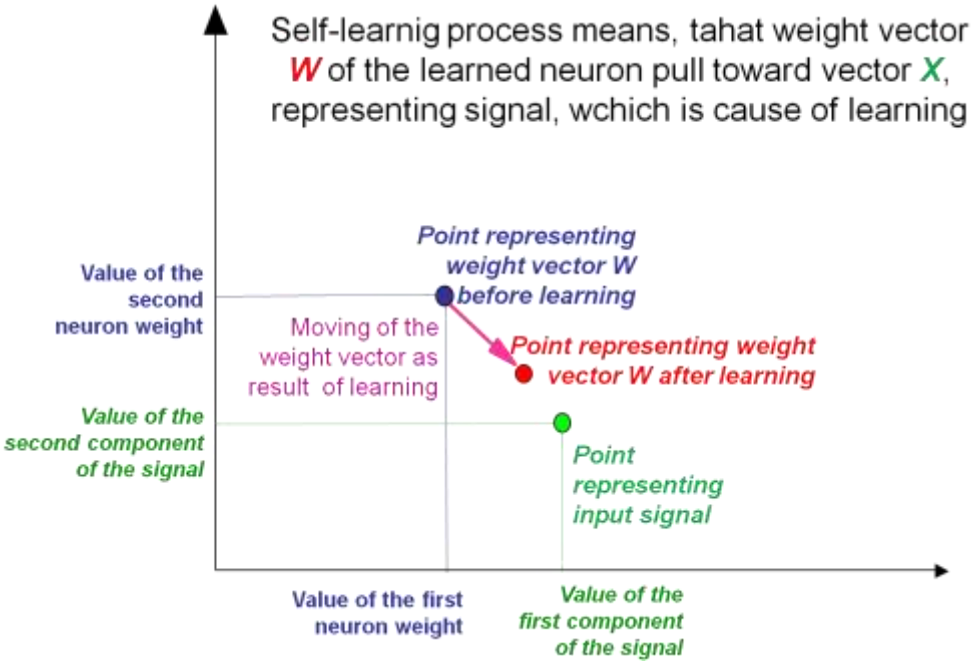


Fig. 10.13. Self-learning process for single neuron

Now, compare fig. 10.14. Notice what follows with it: a winner neuron (marked with navy-blue point) is subject to teaching, because **its** initial weighting factors were similar to components of signal shown during the teaching process (green point). Therefore here happens only amplification and substantiation of natural, „innate“ preferences of this neuron, you could notice this in other self-learning networks. On a figure it looks as if „the winner“ was strongly attracted by an input point, which caused that exactly this neuron has become a winner – its vector of weights (and a point representing this vector on a figure) moves strongly towards the point representing the input signal. Neighbors of a winner neuron (yellow points lightly toned in red) are not so lucky – however regardless of what their initial weights and following it output signals were, they **are** taught to have tendency to recognize exactly this input signal, for which the „remarkably talented“ neighbor turned out to be winner! But to be justly – neighbors are taught slightly less intensively than the winner (arrows indicating magnitudes of their displacements are visibly shorter). One of the important parameters defining characteristics of networks with neighborhood is exactly the coefficient specifying how much less the neighbors should be taught than the winner itself. Please notice that neurons (yellow points), which parameters many times much better predestined them to be taught (they were much closer to the input point) – didn't undergo any teaching during this step.

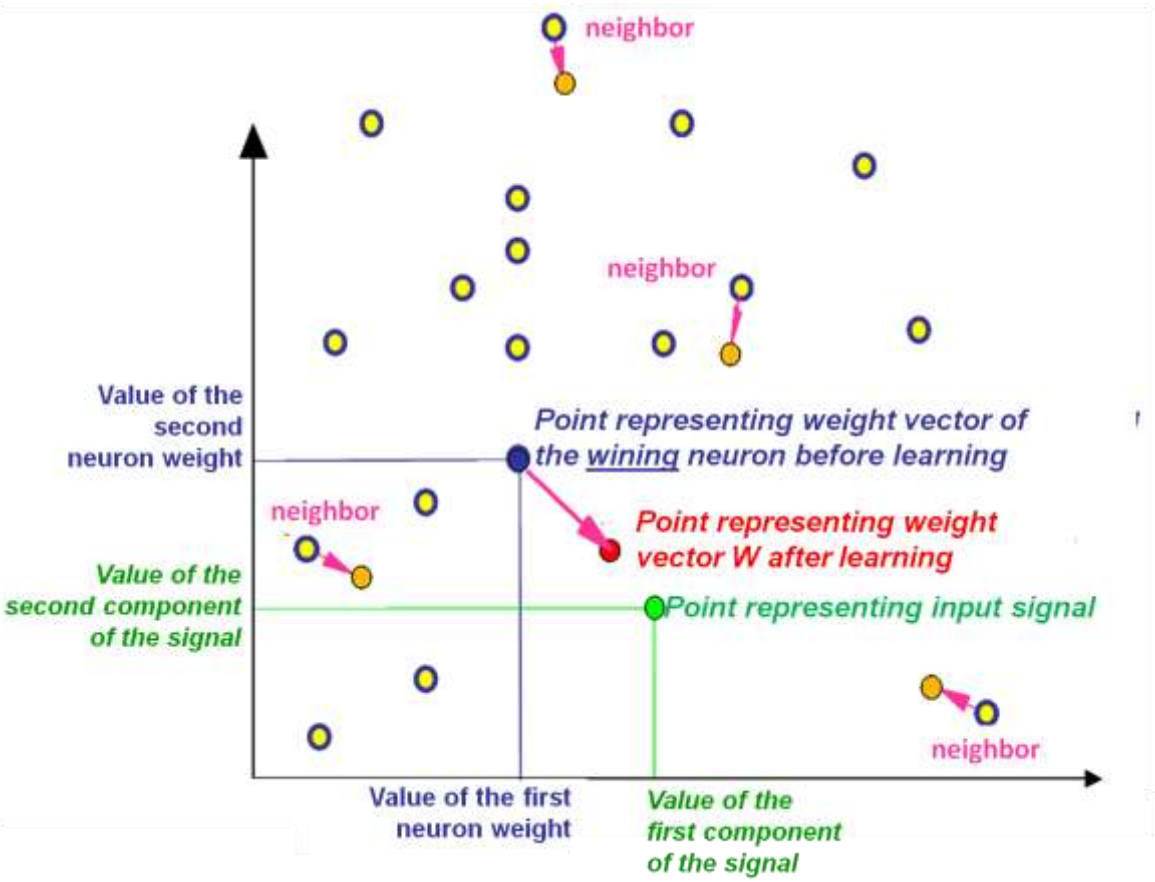


Fig 10.14. Self-learning of the winning neuron and its neighbors

What will be the result of such a strange teaching method?

Well, if the input signals for the network will come in a such manner that there will be clearly existing clusters of them, then the individual neurons will endeavor to occupy (by its vectors of weights) positions in the centers of these clusters, whereas the adjacent neurons will „cover“ the neighboring clusters. Such situation is presented on the fig. 10.15, on which green dots represent input signals whereas red stars correspond with the location (in the same coordinate system) of vectors of weights of the individual neurons.

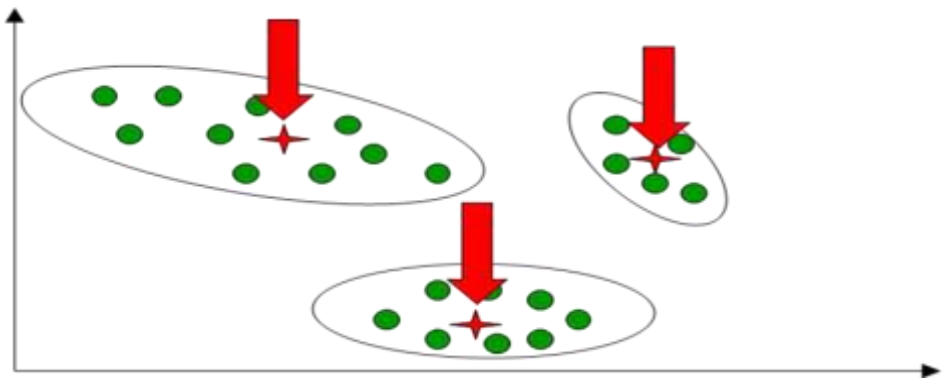


Fig. 10.15. Result of self-learning – clustering of the input data

A much worse situation will occur when input signals will be equally distributed in some area of input signal space, as it is shown in fig. 10.16. Then, neurons of the network will have tendency to “share” the function of recognizing these signals, so that each subset of signals will have its “guardian angel” in the form of neuron, which will detect and recognize all signals from one subarea, another will detect signals from another subarea, etc. Fig. 10.17 illustrates this.

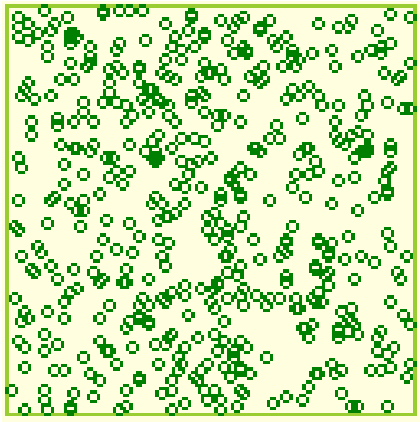


Fig. 10.16. Self-learning using uniform distribution of input data present difficult task for neural network

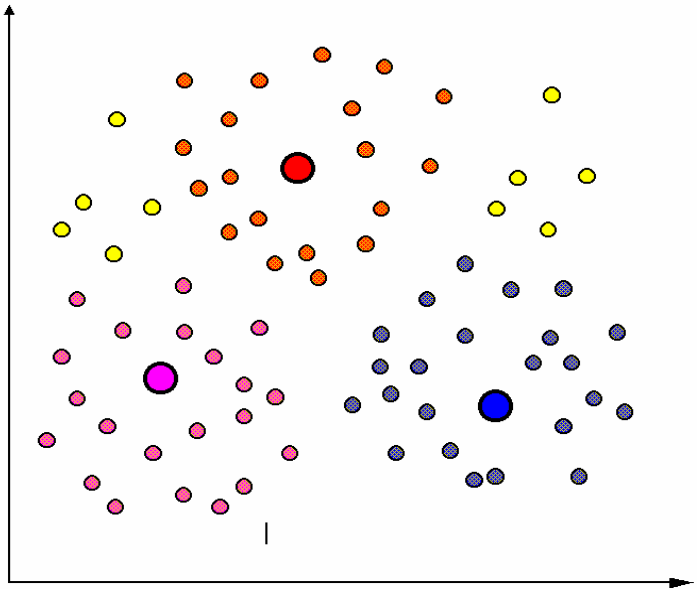


Fig. 10.17. Localization of weight vectors of self-learning neurons (bigger circles) in points of input space, where such neurons can represent some sub-sets of input signals (small circles) in the same color.

While looking at it there is necessary – as it seems – one comment. Well, not immediately may be obvious for you that in case of **randomly** appearing set of points from some area and **systematically** conducted teaching – the location occupied by the point representing neuron's weights will be the central location in the set. But that's how it actually is, as it may be seen in fig. 10.18.

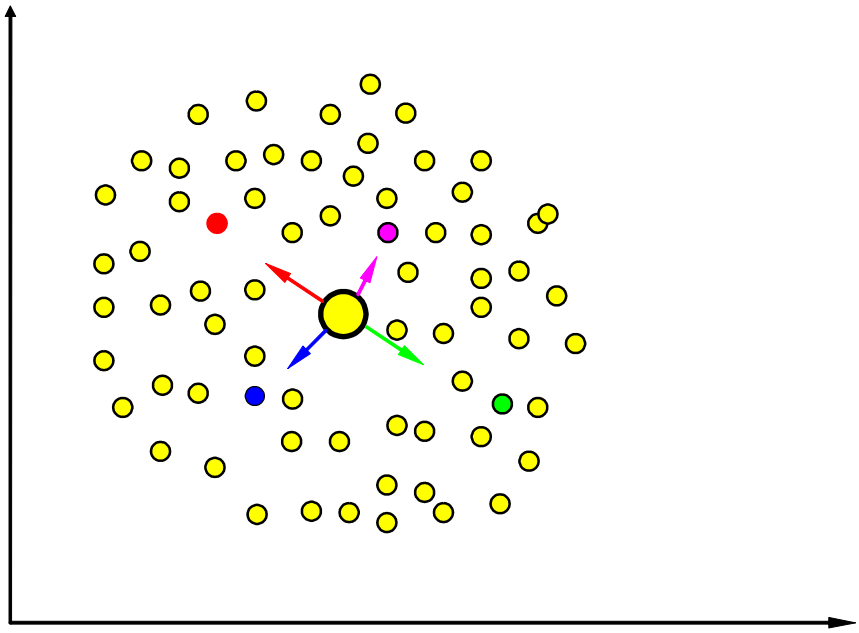


Fig. 10.18. Mutual compensation of pulling from different input vectors, reacting with weight vector of self-learning neuron when it is located in center of data cluster

As seen from this figure, when neuron (represented, as usual, by its vector of weights) occupies the location in the centre of the “nebula” of points which it is meant to recognize, therefore the further course of teaching is not able to move it from this location for permanent, because different points that appear in the teaching sequence cause displacements that compensate each other. To reduce the neuron's “yawing” around its final location, in the Kohonen's networks is often applied a principle of **teaching with decreasing teaching coefficient**, therefore the essential movements associated with each neuron finding its proper location happens mostly at the beginning of teaching (when the teaching coefficient is still large). While points being shown at the end of teaching process very weakly influence the position of neuron which, after some time, fixes its location and does not change it anymore.

Besides this process of weakening consecutive corrections, during the teaching of network there also occurs another process: the range of neighborhood systematically decreases. On the beginning the changes following from the neighborhood concerns, by every step of teaching, many neurons, gradually the neighborhood restricts and tightens so that on the end each neuron is lone and devoid of neighbors (fig. 10.19).

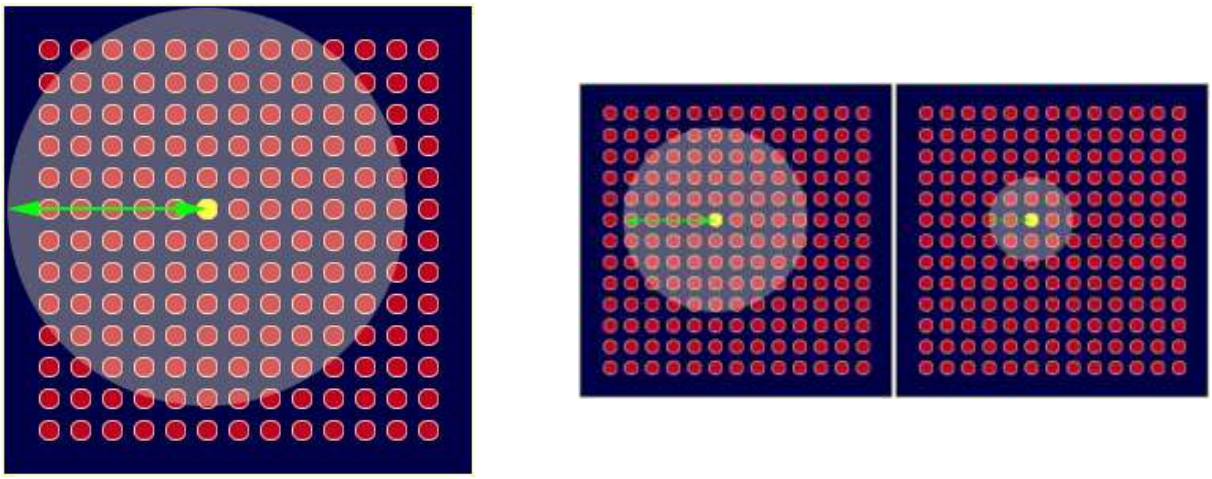


Fig. 10.19. Decreasing of neighborhood area during self-learning process

If you think about all the above information, notice one thing, that after the teaching finishes neurons of the topological layer will portion the input signal space between each other so that each area of this space will be signalized by another neuron. And what more, as a consequence of influence of neighborhood these neurons which you regarded to be adjacent – will demonstrate ability to recognize close – that means similar to oneself input objects. It will turn out to be very convenient and useful because this kind of self-organization is the key to remarkably intelligent applications of networks as self-organizing representations. We were considering this at the particular examples in the first sub-chapters of this chapter.

When presenting the results of teaching of Kohonen's network you will come upon one more difficulty, which is worth discussing, before you contact with a real results of simulations, so that everything was completely clear later. Well, when presenting results (in the form of, occurring during teaching, location change of points corresponding to individual neurons) you must have possibility to watch also what happens with the **adjacent** neurons. In the figure 10.14 you could easily correlate what happened to the “winner” neuron and its neighbors, because there were just a few points and identifying neighbors on the basis of the changed color was easy and convenient. During the simulations you will sometimes have to deal with hundreds of neurons and such technique of presentation is impossible to maintain. Therefore when presenting the activity of Kohonen's networks there is a commonly practiced technique of drawing “map” of neurons positions with marked relation of neighborhood – as in the figure 10.20.

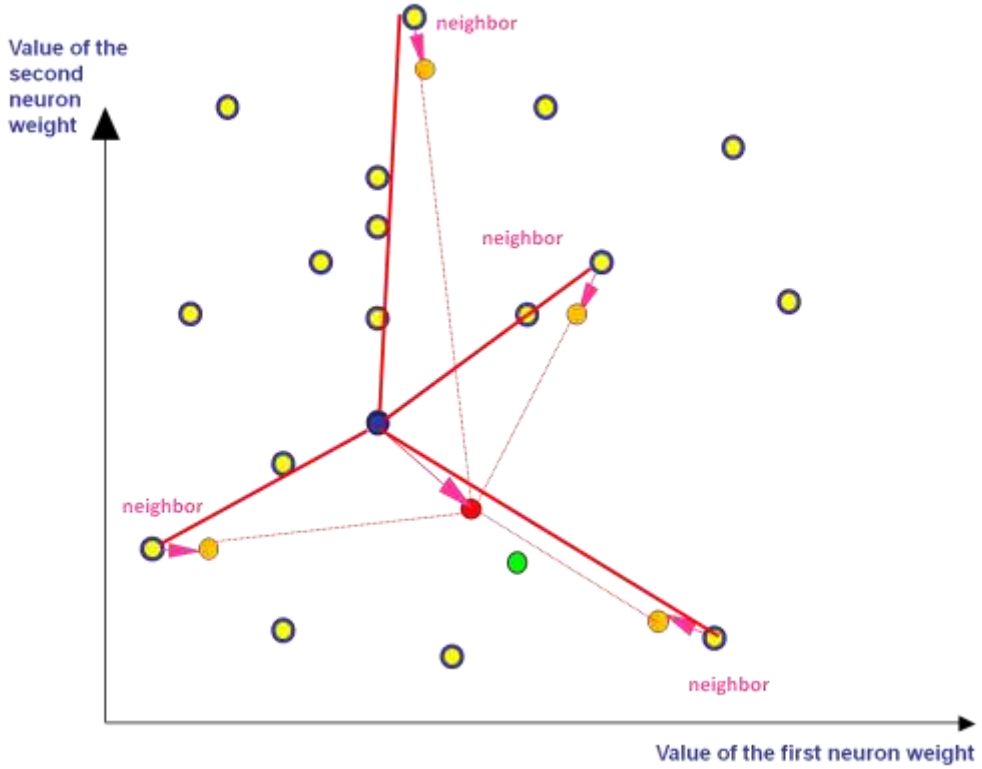


Fig. 10.20. One step of the Kohonen's network learning

In the figure you can notice that points corresponding to the adjacent neurons are presented as connected with lines. If, as a result of teaching process, the points shift – also shift the corresponding lines. Of course this should concern all the neurons and all the relations of neighborhood, but in the figure 10.20. for a maximum clarity I showed only those lines, which referred to the "winner" neuron and its neighbors, while I omitted all other connections. In detail for the full network you will see this, in a while, on the example of program **Example 11**, which I prepared for you.

10.5. What can Kohonen networks do?

Rafał Opiał - oorafal@gmail.com

The program **Example 11** depicts the working of a Kohonen network. But, to let you know how to use it, first a few general comments.

Just after its start, a window will show, where you can set the dimensions of a network: horizontal (**Net size X**) and vertical (**Net size Y**) – figure 10.21. The network size, which you choose depends, naturally, only on you, but to obtain more clear results start the play from not very big networks – e.g. a good idea is to choose a net of size **5 x 5** neurons. Such network is rather primitive and there will be no big use of it when dealing with more complex tasks, but instead, it will learn fast enough so that in a while you will see what is it all about. Then will come the time for networks of greater sizes. In the program code itself, there are no specific limitations for a network size, however the range of inputting these parameters is restricted from **1** to **100**. During the teaching of big networks on a less efficient computer you must sometimes reveal patience, since from previous experiments you know that for finalizing teaching of a bigger network **it is necessary to perform from few to over a dozen thousand steps**. Besides setting the both network dimensions, in the initial window (fig. 10.21), you

can also determine the range of numbers from which there will be drawn the initial values of the weight coefficients (**Range of initial random weights**). For a start I propose you to accept the defaults while in your further experiments you can safely and curiously experiment with another (bigger) numbers since this will show you how the "inborn" abilities of neurons forming the network affect its activity.

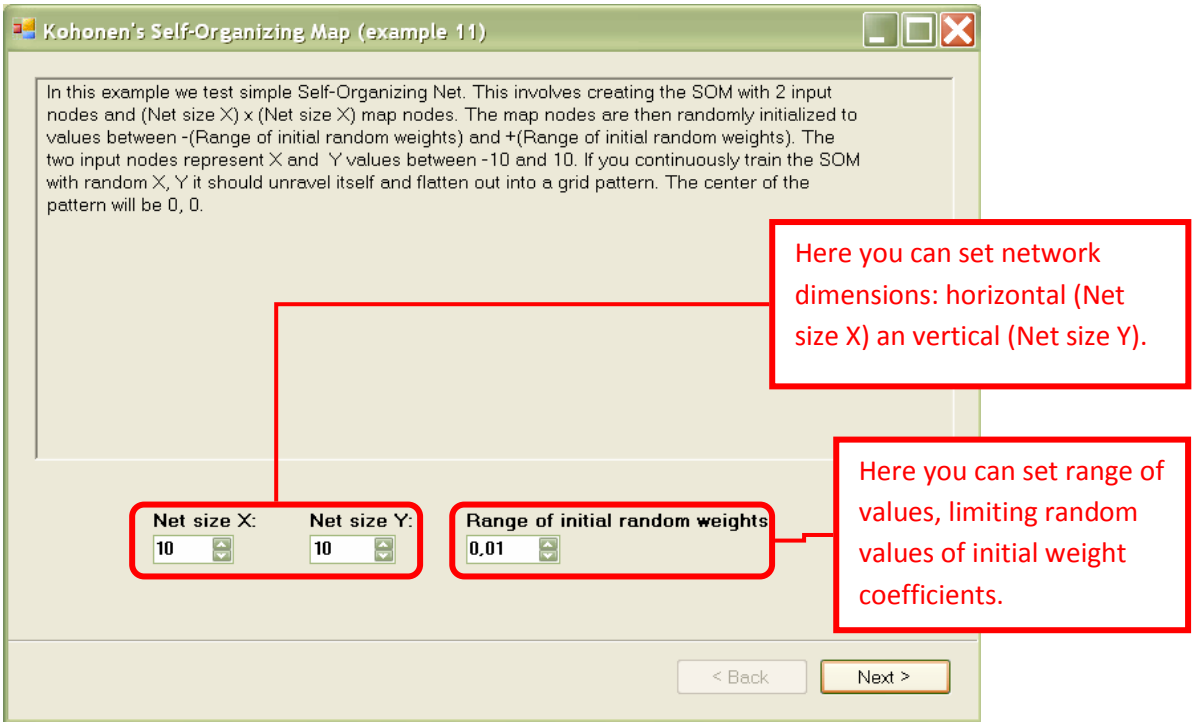


Fig. 10.21. Initial interaction with program **Example 11**

After determining the network dimensions and the range of initial values, by using the **Next** button you will proceed to the next window (fig. 10.22) where you will follow the progress of a teaching process. In this window the location of each neuron in the input signals space will be marked with blue circle while the red lines connecting one circle to another will signalize that these neurons are "neighbors" (according to the assumed rules of binding them together – fig. 10.20).

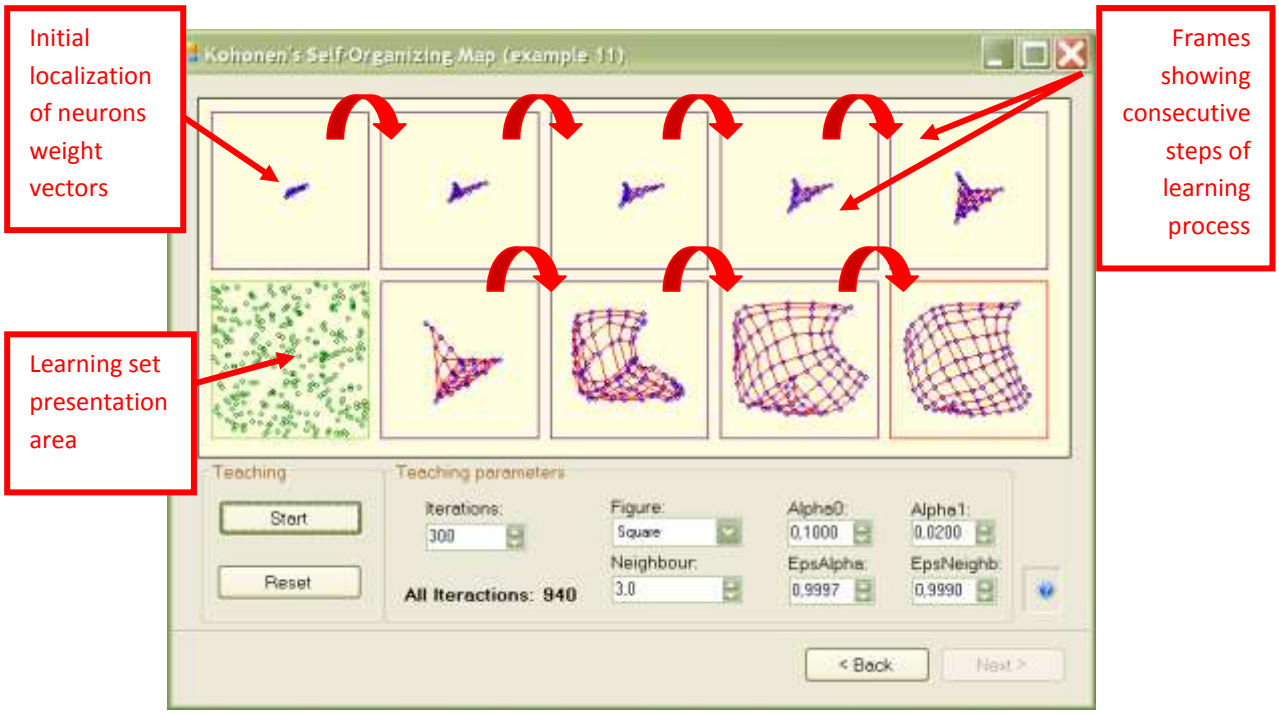


Fig. 10.22. Structure of the **Example 11** working screen

In the point of input signal space, where such circle is located, there is a neuron recognizing and signalizing appearance of a signal from this particular point (and its close neighborhood because the neural networks always tend to generalize the acquired knowledge). During the experiments with Kohonen networks usually there are randomly shown points from a certain subspace of an input signals space. As a result, the blue circles will consequently disperse and spread over the whole input signals space – to be more exact over this particular piece from which will be coming the teaching signals that will be showing during the teaching process. On the other hand these points of input signal space, which will not be shown during the teaching, will not "attract" any neurons. To demonstrate this effect, in the program **Example 11** I prepared three options of presenting the training series: points that teach a network may come from the whole visible area of the input space (this option is called "square"), but them may be taken also from the subarea in the shape of a cross or a triangle. Thanks to this you will be able to become convinced that, truly, a network finds representations of only these input signals that are really shown – in the areas that does not undergo teaching there will not appear (generally) even one circle that represents neuron "lying in wait" in that place!

After determining the number of steps that program must perform (parameter: **Iterations**) you can use a combo box "**Figure**", in a group of training parameters, to determine the shape of an area from which will come the points "recruited" to be shown to a network in consecutive phases of the teaching process. In the group: "**training parameters**" you will find also other parameters of the self-teaching process, but for a start I propose you to accept the defaults.

Now it's enough just to press the "**Start**" button to initialize and proceed the network teaching process. In the window placed in the lower left corner (in the figure it's marked with a green frame) there will consecutively appear nets of points (their count you determined previously in the **Iterations** parameter). All them have the same color because, naturally, there is no teacher that

could somehow classify the input points. But you may notice from which subspace of an input signals space came the signals and it's worth to compare it with the result of teaching displayed at the end.

After pursuing the number of steps, that you selected, the program will display a new map showing the distribution of points symbolizing neurons (as well as their neighborhood relations according to the net topology) on the background of the whole input signals space. Because the previous map is also visible (the program displays on one screen the results of few last steps of teaching process) you can precisely see the progress that the network does during the teaching. The result of the last training step can be easily identified by the red color of the frame. This is necessary when after many steps of the teaching process all of the frames will be taken and the space on the screen will be used "rotational".

At the beginning, before the teaching process will bring some order into it, neurons have rather accidental positions in the input signals space, so the blue circles are scattered with no order and the lines connecting the neighbors toss without rhyme or reason (fig. 10.23).

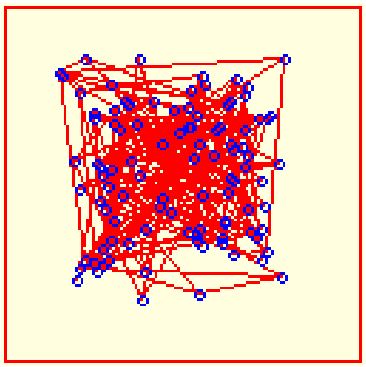


Fig. 10.23. Example of initial neuron weight distribution

As I have claimed in the previous paragraph – the process of setting the initial values randomly, generates the values from the range that you choose. I recommended you to use **small** initial values of weight coefficients in your experiments (e.g. the one that suggests the program: 0,01), so usually the unordered cluster of blue circles symbolizing neurons will occupy the densely filled central area of the input signal space and it will be difficult to see (as it is for example in the figure 10.22). But since I wanted you to see closely how does the initial network state look like before starting the teaching process – before the simulation for which the initial state is shown in the figure 10.23 I have set up a great value for a scope of random initial weights (concretely: 7). I knew that later it will be rather difficult to teach such a network, but I wanted you to see on your own eyes that the initial state of a network (particularly an initial location of lines representing the neighborhood) is absolutely chaotic.

After every display of the neurons location, the program may change the number of steps that must be done before you see a new map of points layout. At the beginning you may use the defaults, and then, when you will "empathize" well with the program working – you may change on your own the number of steps between the consecutive presentations of the teaching results. View the teaching progress on a more impressive way (big steps consisting of few dozens or several hundreds of teaching steps resulting in major changes in the display of network working) or more exact (small steps). At the end of the teaching process, when the "progress" that does the network which is being

taught become really small, it will be expedient to use very big “jumps” (e.g. 100 or 300 steps of teaching) – but remember, in case of a bigger size network you will have to wait a longer while before you will see results!

Our goal is to observe and examine the process of self-organizing in Kohonen Network. First let's investigate how this process goes with a simple cases. If you start a teaching process and a modeling program will start to generate and show to the network points originating from different places of an input signal space (you can follow the process, because in the lower left corner of a program window will be visible an area, where points that are being shown to the network will be displayed – see Fig. 10.22), soon you will observe the move of a blue circles representing weights of neurons. These circles, at the beginning disposed completely chaotic, little by little will be moving to positions equally distributed in the extrapolated area of an input signal space. And what's more there will be an impact on their location coming from the neighborhood relations. You will see that by looking at the red lines showing which neurons are neighbors, that they form a regular mesh. It will look as if the mesh, which consists of nodes (individual neurons) and connections (indicating the neighborhood) is a subject to extension. As a result of it, the vectors of weights of neurons from a topological layer will move to such positions so that each of them will take the position which is a *centroid* (a pattern) for some fragment of an input signal space. This process is exactly described with so called “*Voronoy mosaic*”, but for the accuracy required in this popular book one can assume that different neurons, forming a net, step by step **specialize** in detecting and signalizing different groups of input signals. As a result, after some time, for every input signal that appears often enough, there will be exactly one neuron existing in the net that specializes in detecting, signalizing and recognizing it. In the initial stage (Fig. 10.24) dominates the process in which the random distribution of points and lines is superseded by initial order of points.

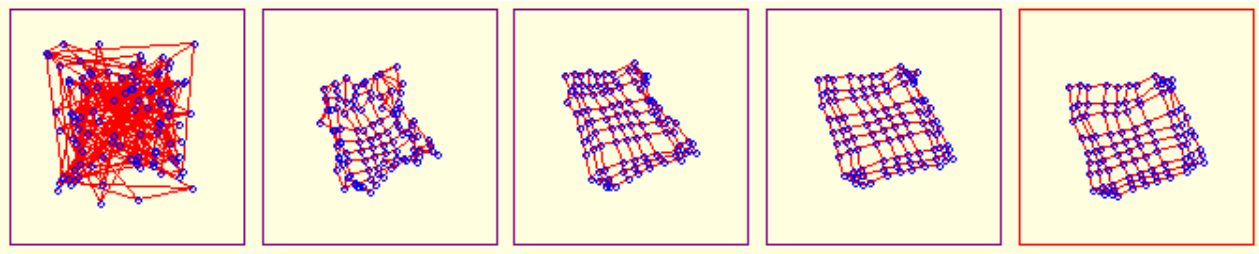


Fig. 10.24. Initial stages of self-organizing process performed in Kohonen’s network

Then the learning process becomes more subtle and tend to make the points distribution maximally regular. The important is that the network creates the inner representations (in the form of adequate distributions of the values of weight coefficients) – only for the subarea of the input signal space, from which origin the points presented to it. And so, you can observe, that in case the input signals come from limited area of input signals (on the figure such area has a square form) – then the Kohonen's network tries to “cover” with the neurons just this square. It happens either if you deal with a network having a little neurons (Fig. 10.25) or for the network (slowly working) with a big number of neurons (Fig. 10.26), as well as in case when the initial distribution of neurons is located on a big area of weights space (Fig. 10.27).

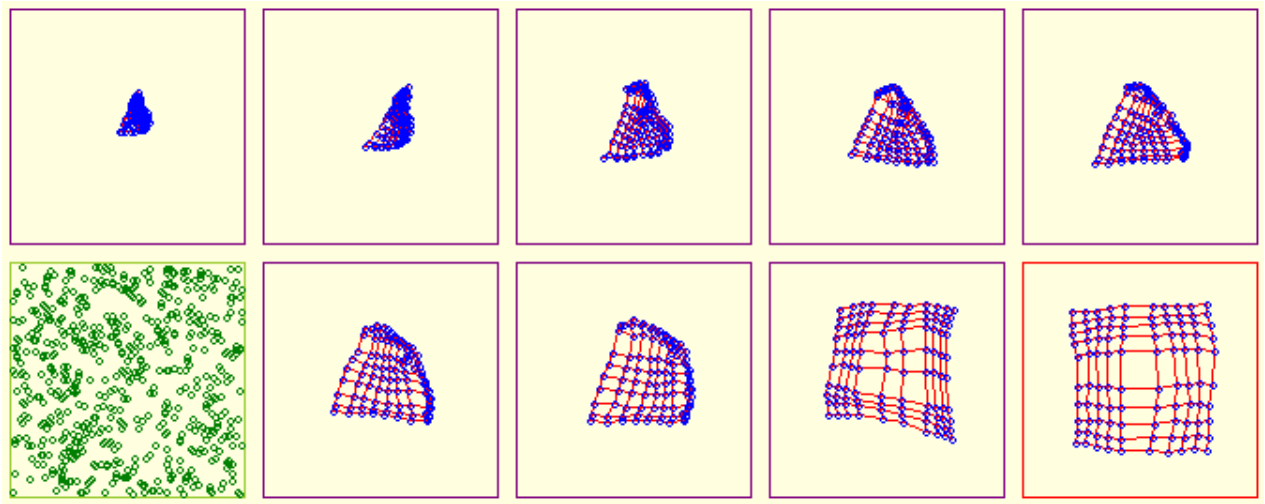


Fig. 10.25. Consecutive steps of self-organization process in relatively small network

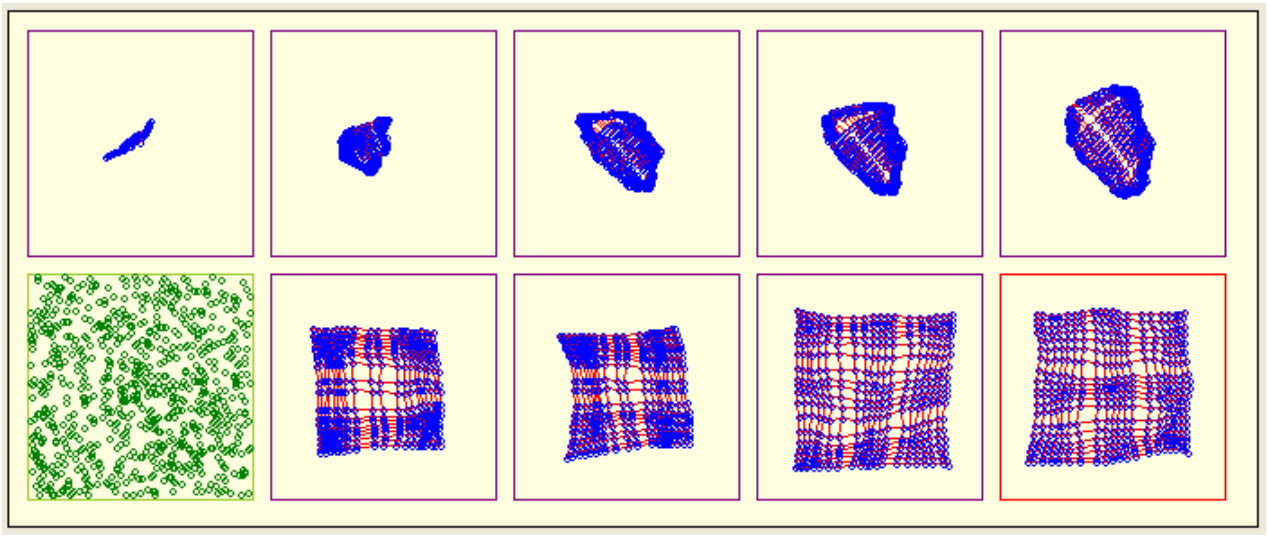


Fig. 10.26. Self-organizing in big Kohonen network

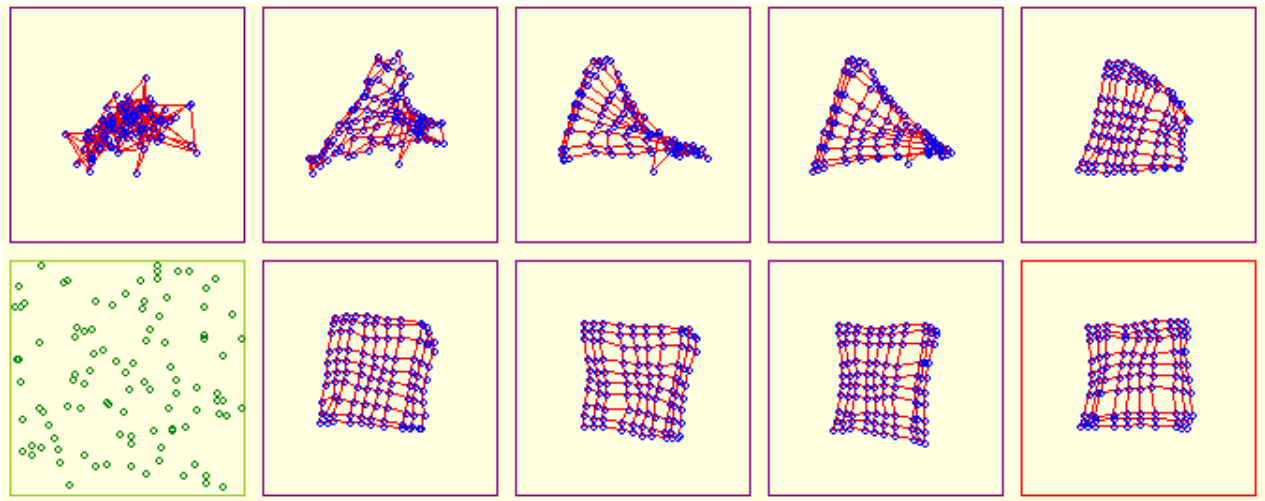


Fig. 10.27. Self-organizing in case wide distribution of neuron weights initial values

Remember that the goal of the self organization process always is to have an individual neuron in the network that detects the appearing of a specific point of the input signal space – even if such a point was never presented during the teaching process (Fig. 10.28)!

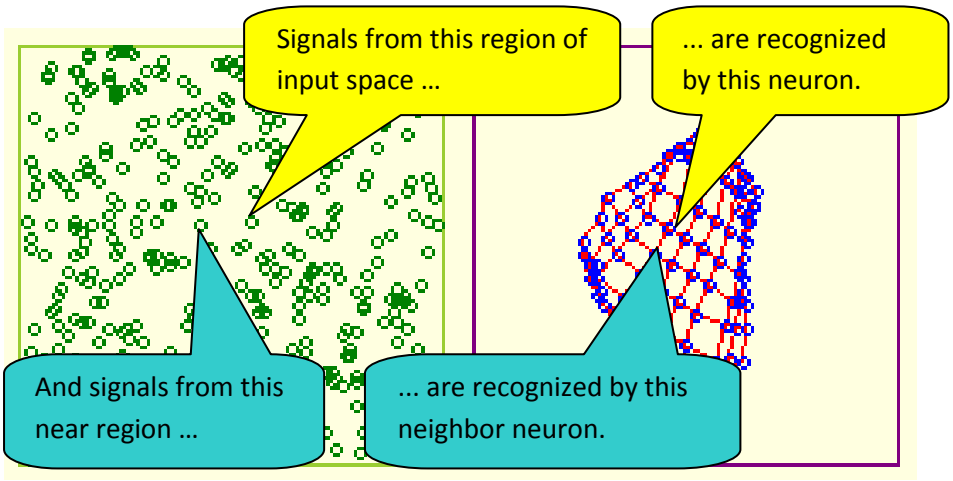


Fig. 10.28. Effect of self-organization. Looking at this picture read comments on yellow balloons first, and blue ones next (from left to right).

10.6. What will Kohonen Networks do in case of a more difficult data?

Rafał Opiał - oorafal@gmail.com

The program **Example 11** has a very rich capabilities that will give you an opportunity to perform – as long as you use your imagination – different interesting experiments. You can, for example, study how the network behavior and self organization processes that happen within it will be influenced by the manner of showing an input data. By using the options offered by the program you will quickly see that if the subspace of the input signal space from which will be coming the values showing to the network, will be limited even more than in case of the square then the self organization process will trend to not create excessive (needless) representations of input data. You can observe it by providing to the network that is being taught the input signals coming only from some chosen subspace of input signal space e.g.: the shape of a triangle (my program gives you such means). You will see then, that all neurons will position themselves to recognize all the points inside this triangle (Fig. 10.29) – no neuron will specialize in recognizing input signals from the space beyond the triangle – because such points weren't shown during teaching, so probably they do not exist and there is no need to recognize them!

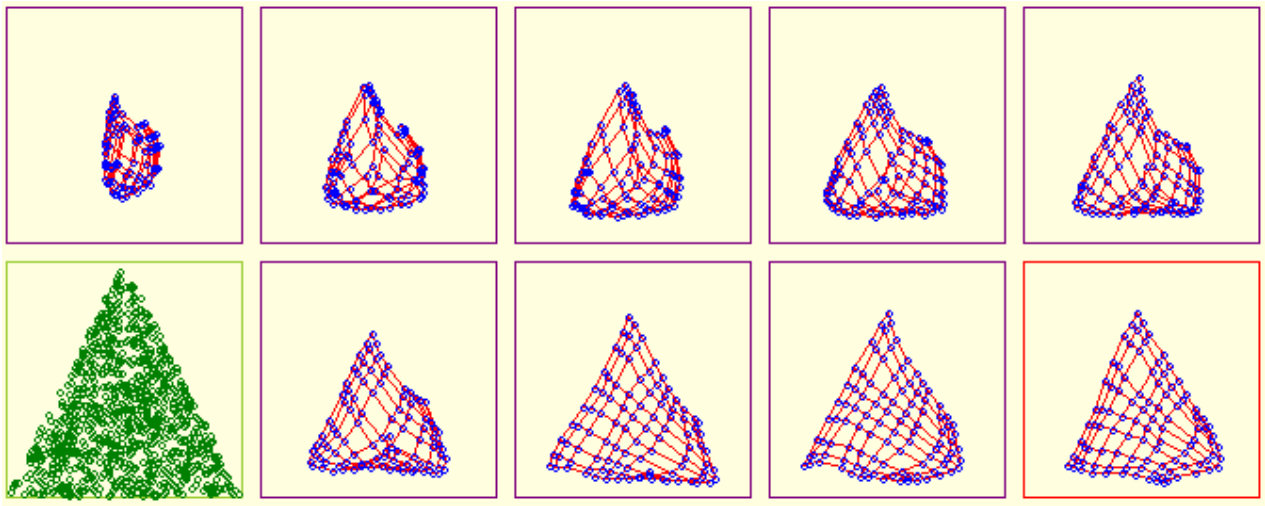


Fig. 10.29. Self-organization in case, when data presented during learning process are taken from sub-area of input space (e.g. triangle sub-area).

A little more difficult for the network is to fulfill similar task in case when chosen subspace of input signal space has more complex form – for example a cross. In such case it may happen that on the beginning of a teaching process the network will not manage to find the proper distribution of neurons (Fig. 10.30).

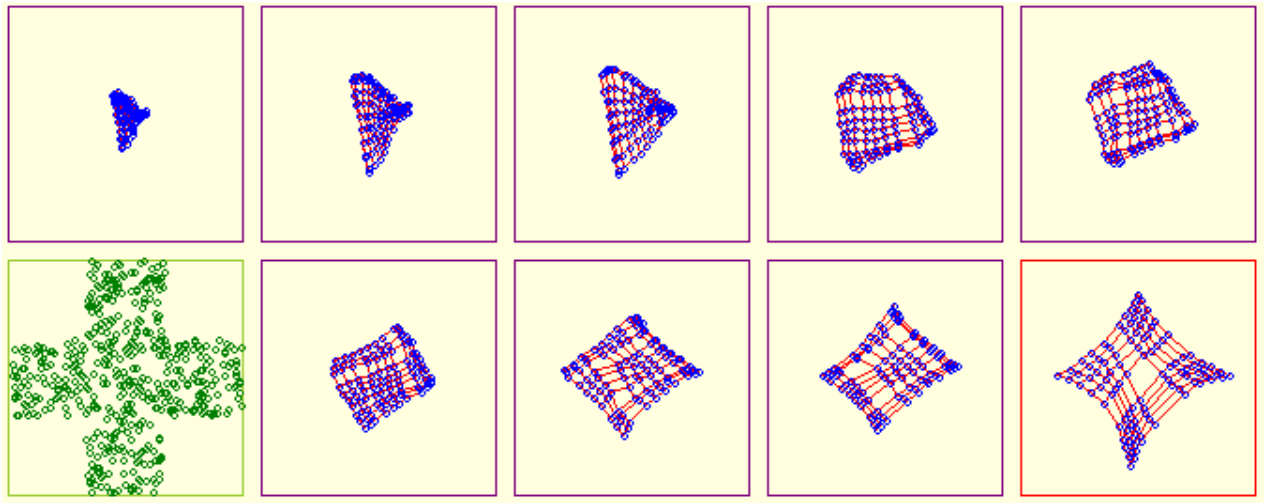


Fig. 10.30. Not very satisfactory result of self-organizing process when input data are randomly taken from input sub-space having form of cross

But usually a tenacious teaching can lead to success also in such case (Fig. 10.31), although this is easier to achieve if the teaching is conducted in a network consisting of bigger number of neurons (Fig. 10.32).

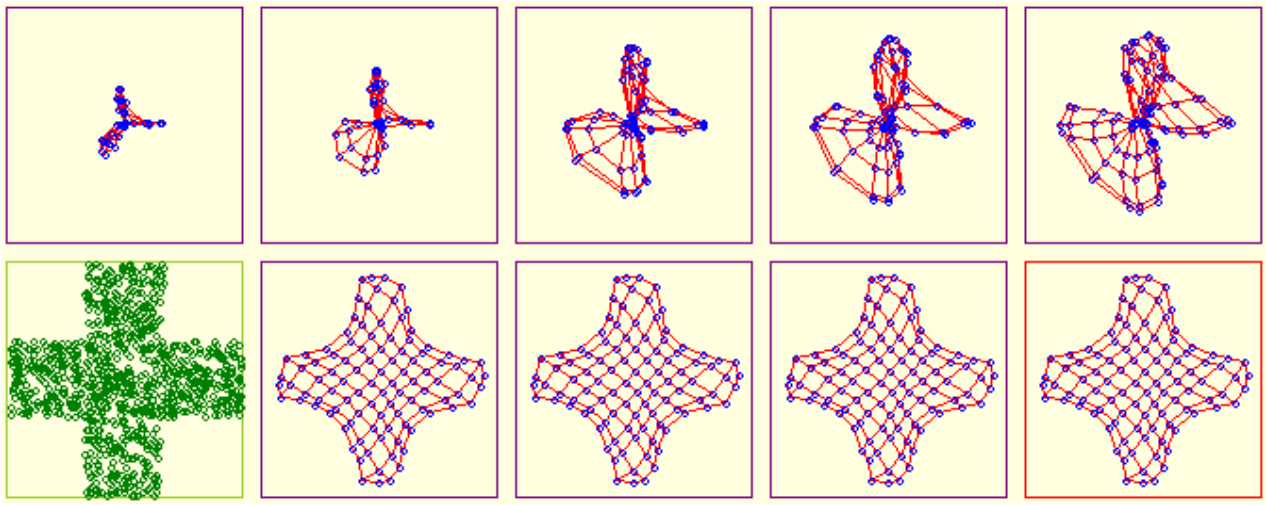


Fig. 10.31. Better (than presented on fig. 10.30) result of self-organization

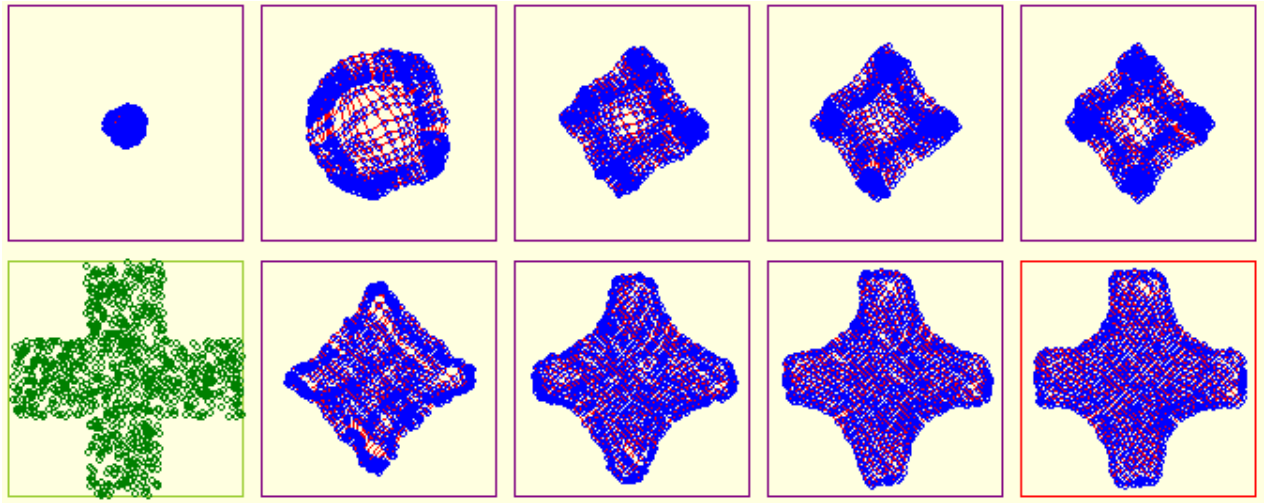


Fig. 10.32. Example of successful self-organization in case of network having many neurons

10.7. What happens in a network with excessively wide range of initial weights?

Rafał Opiał - oorafal@gmail.com

Big initial spread of weight coefficients of modeled neurons is a factor that definitely is not favorable for getting good results of self organization (Fig. 10.33). In such cases often it is being observed a phenomenon of omitting and – despite of long teaching – consequently ignoring by the self organizing network a certain fragments of active areas of input signal space (notice the bottom right corner of the triangle in Fig. 10.34).

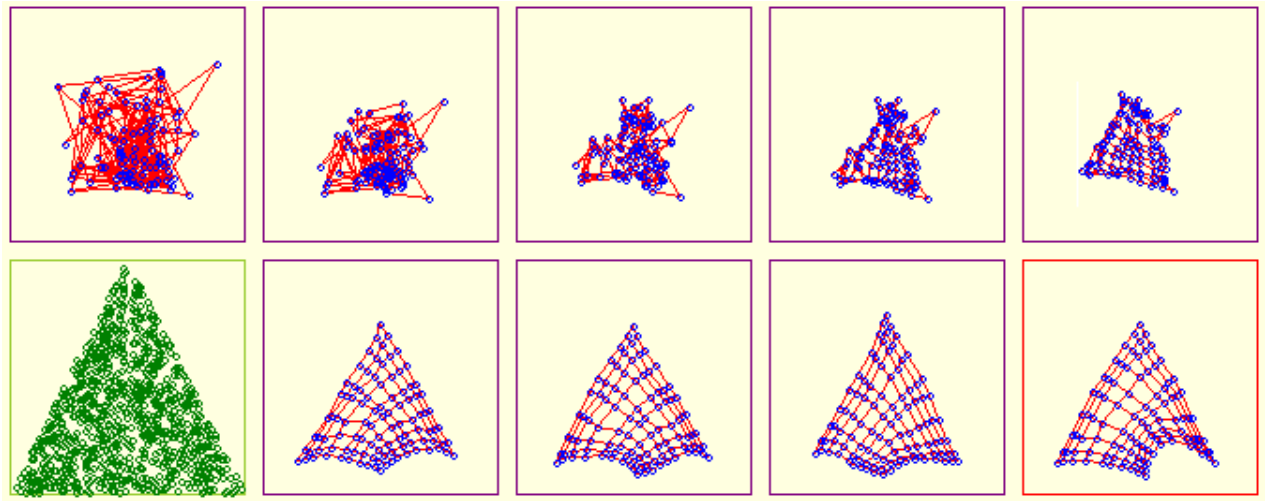


Fig. 10.34. Example of ignoring (by the network) some input data (right lower part of the input space) when spread of initial values of neuron weights is too big.

After you saw the „healthy" functioning of a network, I encourage you much to initiate teaching process with very big values of initial random weights (e.g. equal 5) for some uncomplicated case (desirably for a small network, for example 5x5 neurons, to quickly see the result). Very likely that you will see an interesting phenomenon occurring sometimes in such initially overstocked Kohonen Networks, which is „collapsing" or „twisting" of the network (it is hard to describe – best is to see it by yourself in Figures 10.35, 10.36 and 10.37).

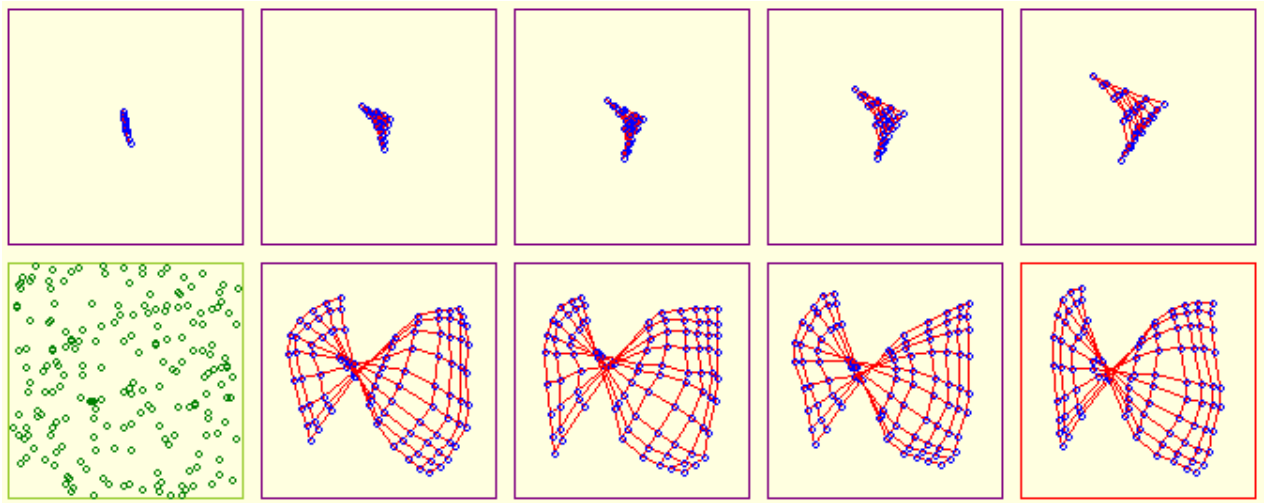


Fig. 10.35. Example of failure during self-organization process.

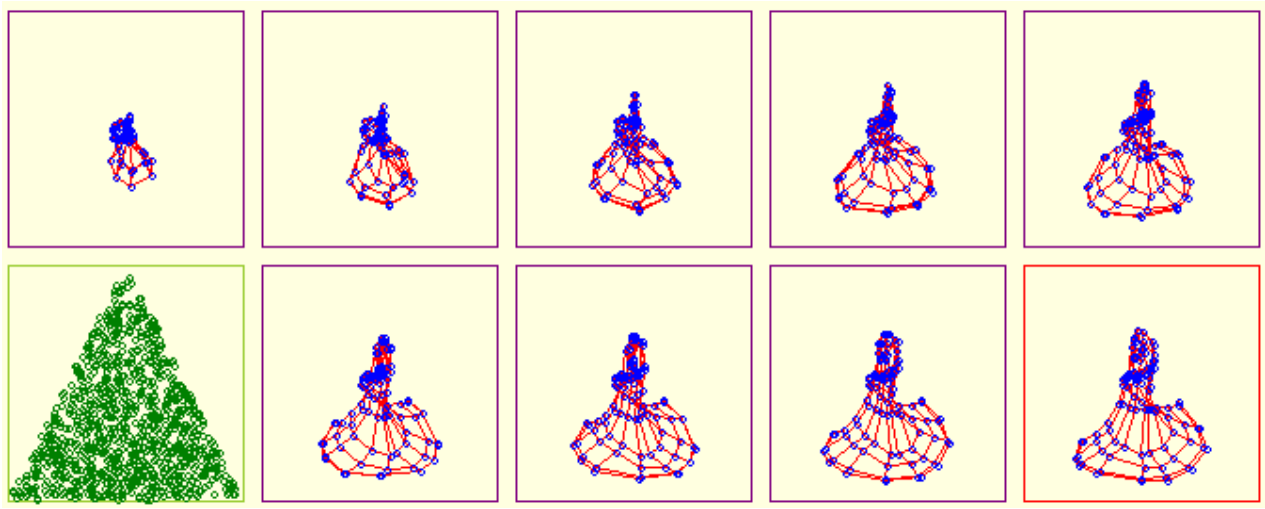


Fig. 10.36. Another example of “twisted” Kohonen network

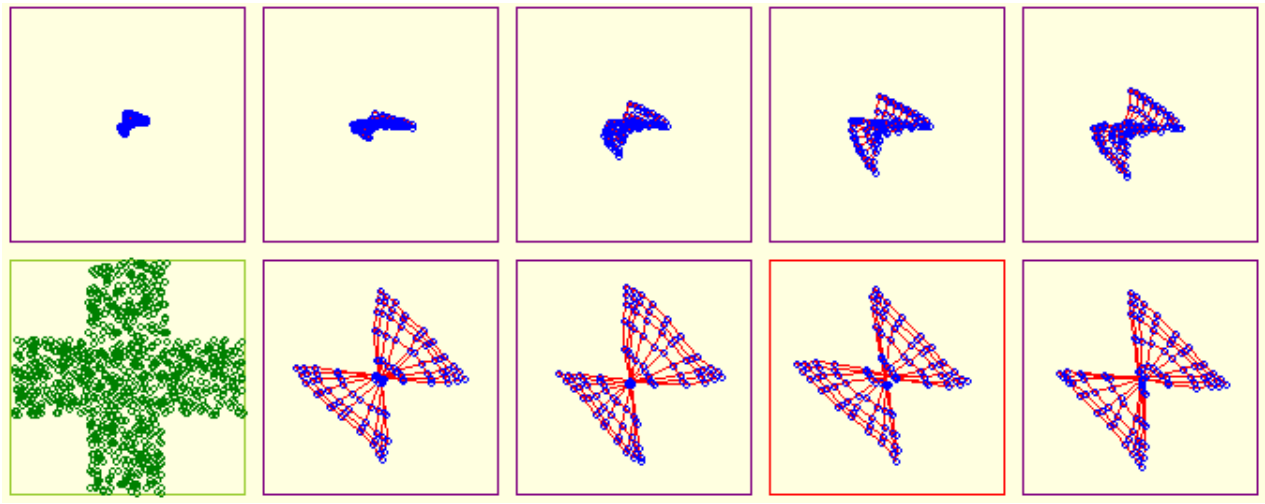


Fig. 10.37. Rare case of “twisted” network learning on data recruited from subspace in cross form

In a free time you may consider what may be the cause of occurring such defect and why the self learning process cannot lead out the network from such a „dead end”?

10.8. Can I change the form of self-organization in the course of a network self-learning?

Rafał Opiał - oorafal@gmail.com

Interesting experiments you may perform with the program **Example 11**, because it allows to change „on the fly” the shape of a subarea of an input signal space for selecting the input data. For example, you are able to start a teaching process with a rectangular shape of a subarea from which come the teaching data, and when the network will almost reach the desired state you may change the subarea shape to a triangle. In the Fig. 10.38 you can see the result of this experiment.

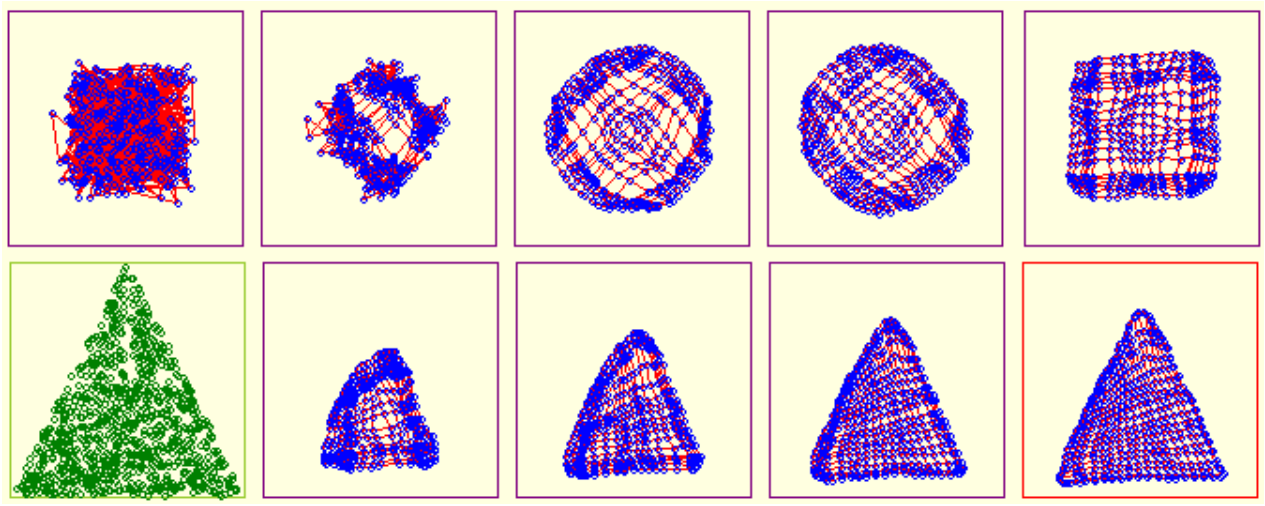


Fig. 10.38. Change of learning goal during the self-learning process

Not always you will manage to perform such experiment in painless way. Sometimes in a final arrangement of a network there will be visible relicts of a previously taught figure. It can be observed in Fig. 10.39, where I have exemplified a course of an especially perverse teaching: at first the network has adjusted to the triangular subarea, then it was forced to recognize an arrangement in a rectangular shape, and eventually, a triangular again.

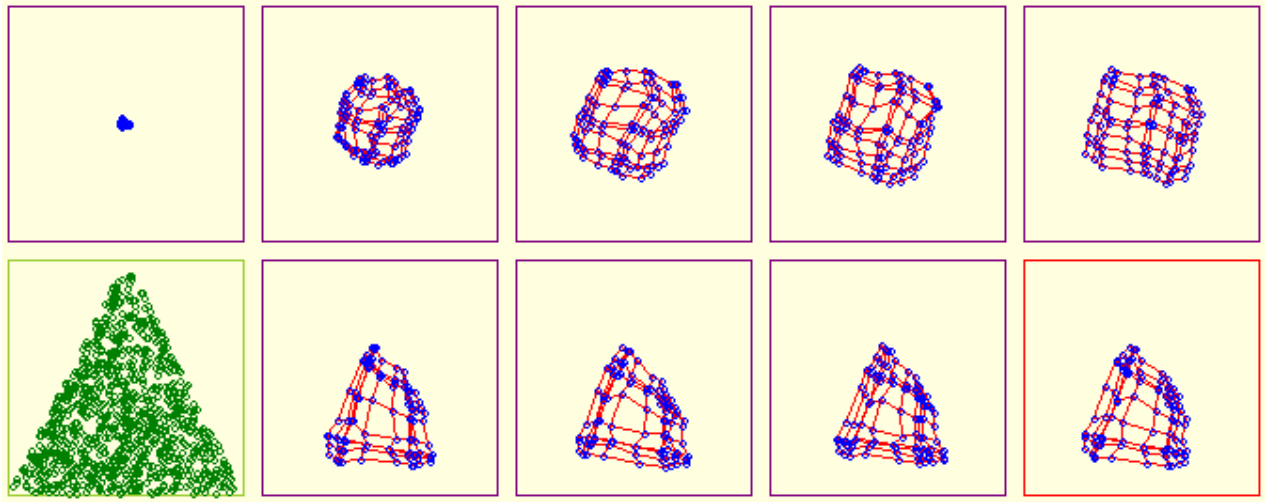


Fig. 10.39. Result of self-learning process with visible relicts of changing goals during the learning process.

As you may learn from the mentioned examples, neurons in a Kohonen network learn (completely by themselves, without any intervention of a teacher!) to map in their internal memory, which is represented by weight coefficients, typically shown patterns of external (input) signals.

10.9. Alright, but what it all might be useful for?

Rafał Opiał - oorafal@gmail.com

I think that by now you have a notion of what Kohonen's Network is capable doing. But still there is an important and actual question – what for can it be practically used? At the beginning of this

chapter I told you already about mappings – for example in robotics – which a Kohonen Network could spontaneously create. By now, knowing how the network works, you can closely analyze this example. So imagine a robot, which has two sensors (because networks that you have already studied always had only two input signals). Let one of these sensors provide information about an illumination brightness, and the other – about sound volume. This way every point of the input signal space will correspond to a one particular environment with a specific set of characteristics – some will be bright and quiet, others dark and loud, etc.

- Say what? Does it remind you of something?

- And very well, exactly it has to remind you!

Now, a robot equipped with a Kohonen Network starts its functioning from observing the surroundings. Sometimes it is bright, sometimes dark, sometimes loud, sometimes quiet – but it turns out that some combinations of input signals occur in the robot's surroundings – and others don't. The robot unhurriedly classifies incoming data, accumulates knowledge about them, specializes its neurons – and after some time it has already trained its Kohonen Network, in which with every practically occurring situation corresponds a neuron that identifies and detects it.

It is worth to think a while, what does it mean. Indeed this what arises in a Kohonen Network that fulfills function of robot's “brain” - is an internal model of the external world. You have such model too. In it, there is a neuron that recognizes Your Mother's face, a neuron related to identification of a way back to your home, a neuron that recognizes your favorite cookies and one that signals a presence of hated neighbor's dog, which has bitten you twice already. For every sensory perception that you recognize, for every situation that you know – you have a model in your brain that detects and recognizes it. An outstanding polish neurophysiologist professor Konorski has associated these internal models of particular fragments of the outside world with separated parts of human brain and called them gnostic units.

Practically whole your perception, whole ability to know and recognize the condition of a surrounding world is based on fact, that in your brain during years of communing with different situations, you have created a ready patterns of appropriate perceptions – exactly these gnostic units. Signals that are provided by your eyes, ears, nose and the other senses serve in this situation usually to activate a proper gnostic unit, that means to select and run the exactly one of thousands models stored in your brain, that corresponds to the encountered sensation or situation that you are. Thanks to that – and as it turns out – only because of that, your ability to recognize sensual expressions is so fast, efficient and reliable. But if your brain lacks such ready, prepared earlier model, then the perceptual situation becomes difficult, and orientation in the new situation is often very delayed and often unreliable.

There exists a hundreds of evidences for that. To not extend this topic too long and let You as soon as possible play again with a neural network – I'll mention only three. The most important evidences that the shown above internal models of the external world really exist and are indispensable, were gathered while performing experiments on animals. It was performed, for example, deprivation of young cats1.

It consists in that when a young cat starts seeing (as is well known – kittens born blind and start seeing after few days) – the researcher let them to see only some given geometrical pattern – e.g. vertical lines. On every occasions when a cat could see anything else (e.g. during feeding) – light is being turned off. After a month of such training a cat was returned to live in a normal world. It turns out that a cat having healthy and fit eyes – behaves like a blind. He is not able to notice an obstacle, bowl with milk or even a human. His brain holds only models of single geometrical figures, it does not have patterns of a chair, bowl or other cat. His perception is completely disrupted and it needs a long time of learning so that the cat regain the ability to see normally.

About the fact that similar phenomenon occur also to humans show information of anthropologists who described tales of Pygmies. This is a tribe of African midgets which the natural habitat is a dense jungle, where there is not possible to see far away. Now, when these people were led to an open space, they completely lost the sense of direction. The simplest phenomenon related to looking far, trivial for each of us, such as changing a perspective related to e.g. approaching of some animal – were treated as a result of an unintelligible and powerful magic (yet a while ago the zebra was smaller than a dog and now it has grown to the size of a horse!). In their brains lacked models regarding the perception of visual phenomenon on an open space.

Sometimes about the meaning of internal models of objects from a world recognizable by you, you can see on your own example – if you pay attention to simple and apparently obvious phenomenon. For example you easily read news in your mother tongue, and since you are an computer expert, probably in English as well, or maybe in German, French, Spanish or another. With each of these languages, if you know it well enough, you need just glance – and in your brain the letters turn into words, from the words form notions, from the notions knowledge... It occurs because in your brain for many years of learning were formed models of letters, words, sentences and whole pieces of information, because of that while reading you just recall them and they appear immediately – ready to use.

Different situation appears when you encounter an unknown word in a text, or even you try to read a message in an unknown language. This time with a much effort you syllabize the text and it takes you a lot of time to spell it, saying not about the total inability to understand a meaning contained in the given statements. Easiness and fastness of reading disappear irrevocably, because in your brain there are no ready patterns, there are no models for these words, sentences and messages.

I could extend this thread and show that in case of inscriptions formulated in an unknown language (e.g. Hungarian), but using a well known alphabet – it is possible to utilize models of letters existing in brain, thanks to that you can remember an unknown inscription and when you meet a translator you can ask him to translate it for you. However if in your model of the outside world you do not even have patterns for letters (e.g. if you encounter an unknown inscription in Japanese or Iranian) – then you will not even fail to understand the matter of it, but you'll even have much difficulties to remember it and ask for help your familiar translator. Simply, your brain in this case will not have any useful gnostic units.

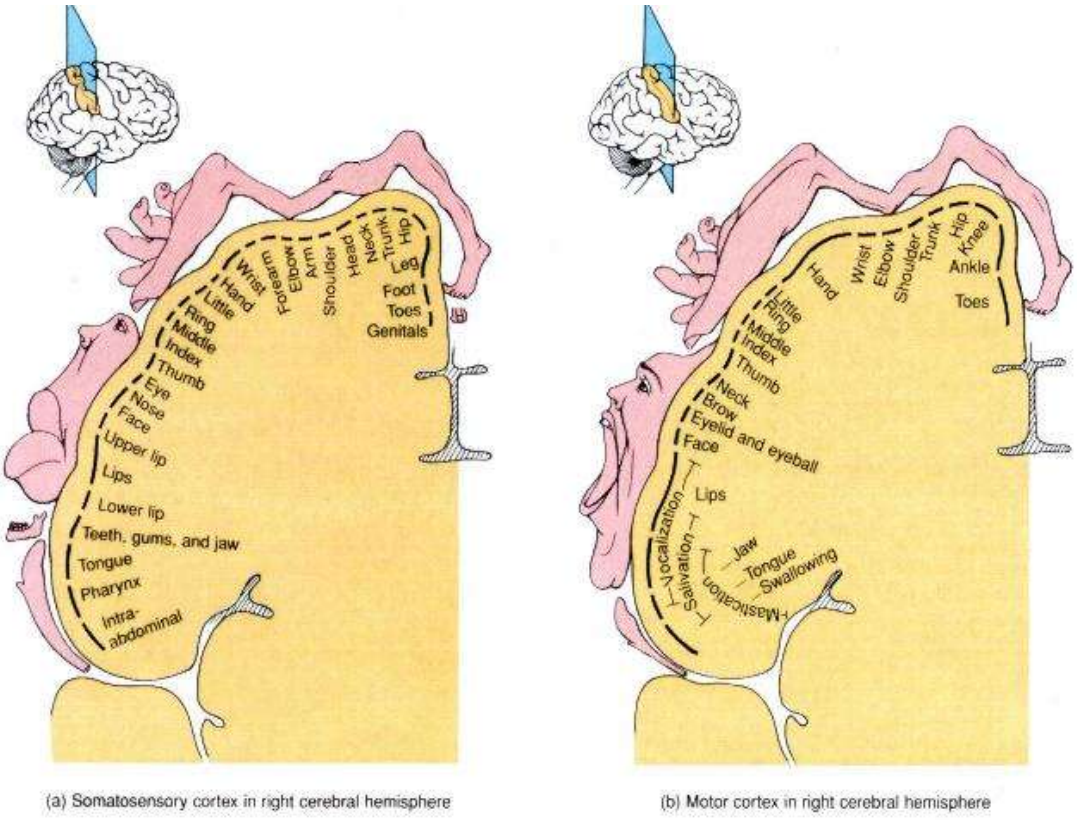


Fig. 10.40. Internal representation in the brain somatosensory stimuli regions and motion control regions (source: http://ionphysiology.com/homunculus1.jpg, taken may 2012)

¬In your brain there is also another representation which creates a map of all your body on the surface of cerebral cortex in the area of ganglions called *gyrus postcentralis* (Fig. 10.40). What's interesting: the map does not resemble exactly the shape and proportions of your body! On the surface of a brain, for example palms and face occupies much more space than a whole trunk with limbs (Fig. 10.41)!

It results exactly from the fact I mentioned above in the context of artificial self organizing neural networks: controlling the moves of palms and face (mimic, speech), and also reception of sensory stimuli involves more brain cells because it is done often.

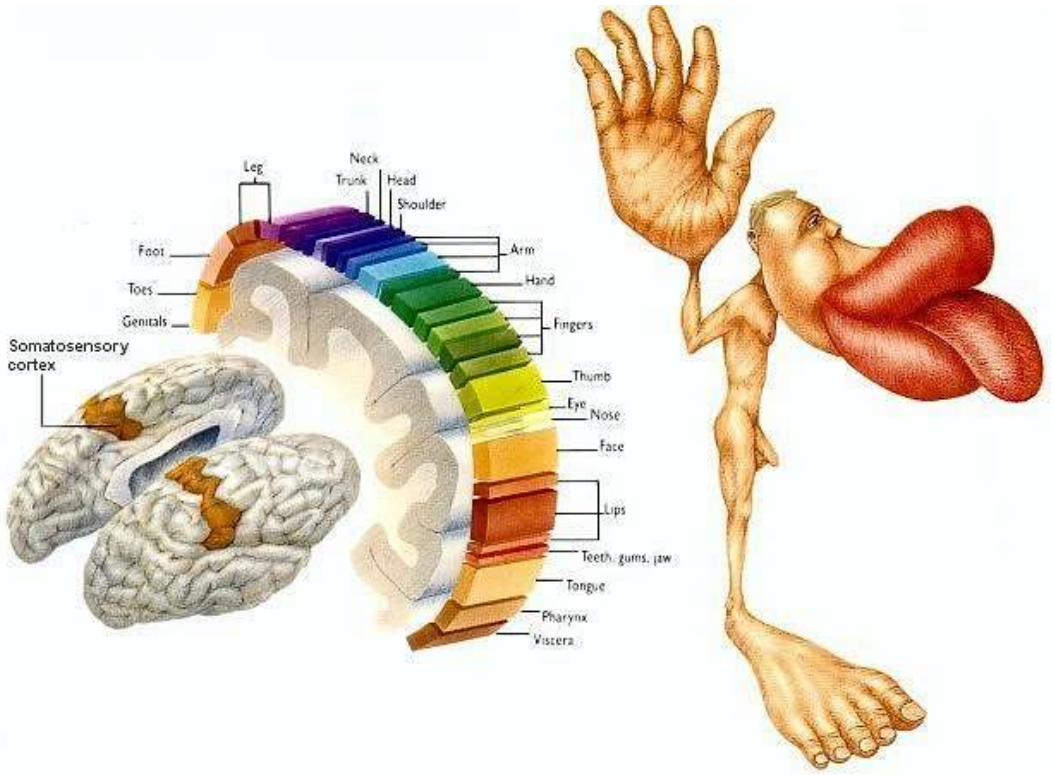


Fig. 10.41. Homunculus (Image courtesy: www.gmail.com). Each area of body has proportionate sensory neural area.

Excuse me that I wrote at length on the examples. I cared, though, that you understand and remember how great and important meaning have patterns of assimilated sensations early prepared in brain in the course of seeing (or any other perception like: hearing, taste or even touch). And such patterns are being created fully spontaneously in neural networks during the process of self organizing.

Let's come back now to the example with the robot which I started few sentences earlier. Every neuron in this robot's Kohonen's Network, recognizing one of the occurring (and only the occurring!) states of an external world, serves the same function as, the described above, gnostic unit in your brain. Therefore the self learning process of a Kohonen's Network creates a specific set of gnostic units inside the robot's controller, which are a set of models for every situation in which the robot may be. Such model is very important because having a model one can classify every input situation (following from the signals provided by sensors). Whereas after classified an actual situation to one of prior defined classes the robot may adequately adjust its behavior. In the simplest case you may define what a robot must do in specified situations – e.g. you may state that if a situation is normal (quiet and not very loud) the robot must continue moving forward, if light is turned off it must stop, and when it hears a noise it shall step back, etc. The important thing is that you do not have to state a detailed directions what the robot must do in every situation. Based on a neighborhood of neurons detecting similar situations, you may stop after giving a clues for some selected situations, and the trained Kohonen Network in the robot's brain will figure out itself what to do. Simply, if it finds that for some specific perceived situation, it does not have a ready recipe for how to behave, it will figure out by itself (based on the neighborhood structure in a neural network) that, from a set of different states of a network, for which there was a proper behavior defined, the closest is some specified

another situation. If for this situation it was saved in a memory that some action is necessary – then for currently recognized similar situation it is advisable to do the same or something similar. For example if for the respective analogical situation it was advised to quickly go forward – in all not defined by the user “neighboring” situations there can be applied “go forward” but with a reduced speed.

By that means, if for just few model situations you will define what and when to do – a robot will be able (better or worse) to behave in every possible situation. It is worth noting that it refers not only to the objects that occurred during the self learning of a network. During self learning of a Kohonen's Network, you deal – as it is usually with neural networks – with an averaging and generalization process.

Regarding to the averaging process, it is worth to note another fact. Well, during the teaching, there may be shown objects (environments) that slightly differ one from another, that means they will be characterized by a certain dispersion – and yet it will be remembered for them (in a form of a set of weight coefficients for certain neurons) certain resultant, averaged pattern of a “typical input signal” which the network will designate to itself (completely by its own!) during teaching. There will, of course, exist many of these typical sample signals (and sample environments related to them) – exactly that many as many neurons is contained within a network. However there will be much lesser of them, than possible environments, because with freely changing parameters that characterize situation in which a robot can find itself – number of possible environments is infinite!

However, thanks to generalization, during the “exam” (that is during a normal exploitation of a robot) the network may find itself in an environment characterized with such parameters that have never been shown during teaching. But every neuron of a network, even if it contacts for the first time with some signals, tries to qualify them to such a group, for which a pattern created during teaching is the most similar to the actually considered signal. It causes that knowledge gained by the network during a learning process is automatically generalized during exploitation – which very often gives a perfect results.

The example with a robot, described above, that, thanks to a Kohonen Network adapts its behavior to an environment appealed to your imagination (I hope!). But on a daily basis people do not need to many intelligent mobile robots, so if you had to live from building Kohonen Networks for such robots – you would not afford a villa with a swimming pool and an exclusive car – which you certainly deserve. Unfortunately it often happens that even the most brilliant technical systems does not earn money to their creators – if there is no buyers that will to utilize them. Maybe you already created in your brain a model of this situation? If so, then let's think – who and for what reason could possibly use a Kohonen Network in reality?

I will not summarize all the possible ideas here, because there is too many of them, but some of the most interesting ones I will – just to give you a starting point for your own conceptions. The previous paragraph talked about money. And money, as it is known, is in banks, and banks sometimes are robbed...

No, I am not convincing you to build a mobile robot controlled with a Kohonen Network, which when were in front of a cash deck will say with a metallic voice: Hands up! This is a robbery! Give me a dough! I induce you, though, to think about a possibilities that lie in Kohonen Networks regarding the

protecting banks against a theft. But not against a theft as seen in movies – with a masked bandits, a shooting and an escape in a car, a very fast car on winding streets. Such thefts occurs (fortunately) rarely, and Police have its methods to prevent them. However a true problem of banks, are nowadays thefts done in a white gloves – for example by extorting credits and not paying them off.

Bank lends money to exist, so it cannot throw out the door everybody who comes with a credit application – although it would guarantee a hundred percent security. However some borrowers do not redeem money they borrowed – it generates losses, usually much bigger than famed armed robberies.

How recognize to whom money may be safely lent and to whom not?

It is simple – you need to have a model of a honest entrepreneur who will build a fair prospering company for a borrowed money and will pay off the credit and interest, and also you need a model of a trickster who wants only to rake in and hit the road with money. Unfortunately people are very different and different are life and economical situations, so there must be thousands of models of a honest businessman, and at least the same number (or maybe more?) of tricksters and embezzlers. No man will manage to complete such a task.

But what for are the abilities of a neural network? Enough to think over well, what data to introduce to it, and how to perform self learning, how to interpret then results – and after selling this brilliant system to few banks you can go for a deserved vacations on a sunny beaches of Caribbean.

- What? To little information? Shall I describe more accurately how to do that?

But do you know what a crowd would be on a coral beaches of Puerto Rico if I gave here all details and everybody who reads this book would become a millionaire? You shall use your head a little on your own…

10.10. How the network can serve as a tool for transformation of an input space dimension?

(Translation by Rafał Opiał, rafal.opial@op.pl)

After that short trip to blue seas and humming palms, lets now come back to next scientific problem. An unique characteristic of Kohonen Network is that within its responses there exists a **topological representation of an input signal space**. Of course it applies for neighborhood and its consequences. In figures that you saw above, and on those you will see during your own usage of the described program, there are blue dots representing neurons, connected to each other with red lines. These lines, as you know, connect neurons considered as adjacent. At the beginning these lines – similarly as dots itself – are distributed randomly. During teaching you will notice thou that a network during learning will develop a **regular net** consisted of red lines. What, in fact, the net represents? Well, it is an expression of rule consisting in that **adjacent neurons** will tend to signal and detect **adjacent points** from an input signal space! This way the order coming out of the fact that some points representing input signals are close together will be transposed to a network in which adjacent points will be signaled only by adjacent neurons.

In all the figures seen so far, two notions: closeness (similarity) of input signals and adjacency (neighborhood) of neurons in a network could be easily associated because input signal space (and of course weights space) was two dimensional (as in Fig. 10.13 or 10.16) and during that time also a network topology was two dimensional (as in Fig. 10.9 or 10.11). So, there existed natural correspondence between such notions as “an input signal lying higher than the previous signal” (in this sense it had a higher value of a second component) and “an input neuron lying higher than the previous neuron” (in this sense that it lies, according to agreed numbering – in the previous row (Fig. 10.42).

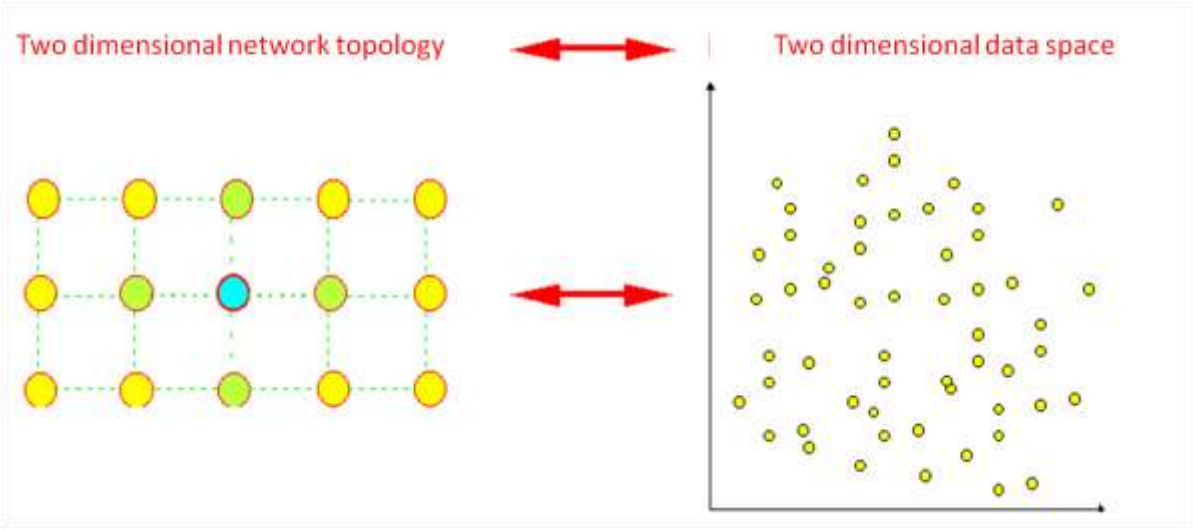


Fig. 10.42. Example of same dimension of network topology and data space

Such situation is not the only one possible. One can easily imagine a one dimensional network which will learn to recognize two dimensional signals (Fig. 10.43).

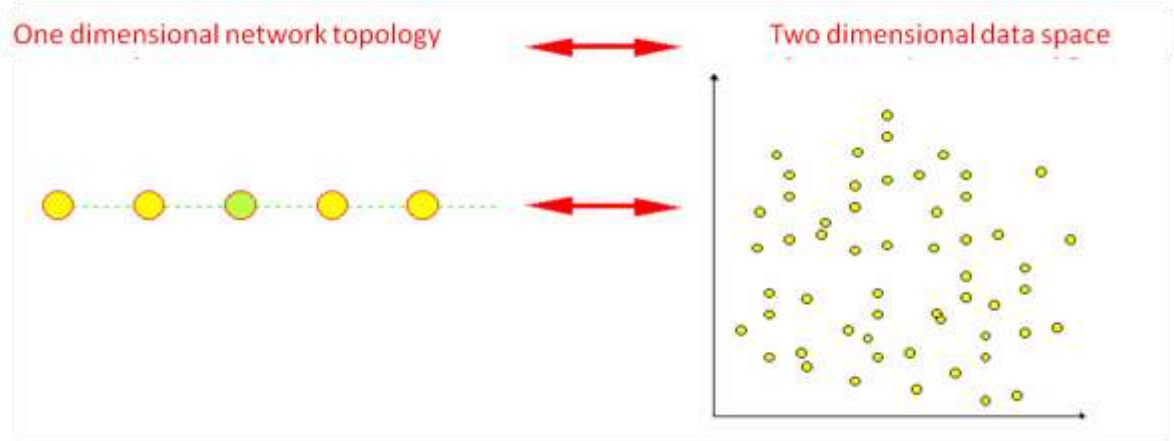


Fig. 10.43. Example of different dimension of network topology and data space

You may study how behaves a network that makes such a conversion of two dimensional input signal space to one dimensional structure of a network, because my program allows you to create **one dimensional networks** (chains of neurons).

Such untypical structure of a network you may obtain by giving one dimension of a network (the best the first one) equal 1. The second dimension is then worth to define rather big, for example 100,

because then the network behaves in an interesting way. It learns though pretty fast, so there is no need to limit yourself (in this case).

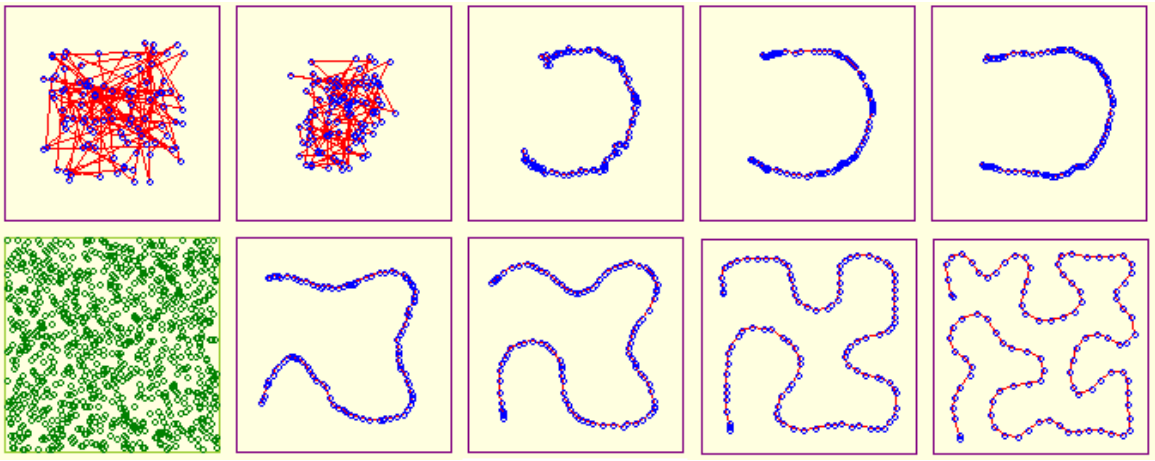


Fig. 10.44. Mapping of two-dimensional input space into one dimensional neural topology

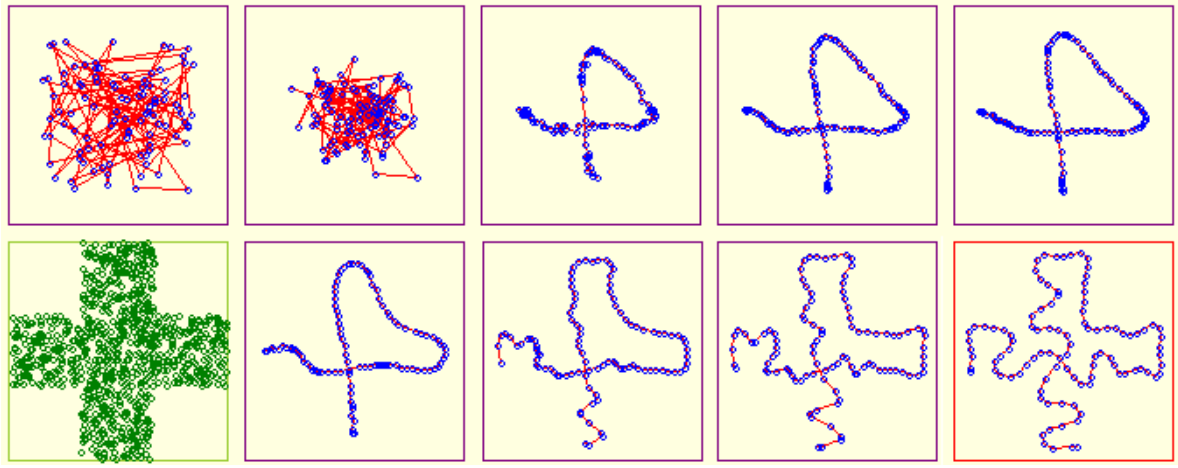


Fig. 10.45. More complicated example of mapping of two-dimensional input space into one dimensional neural topology

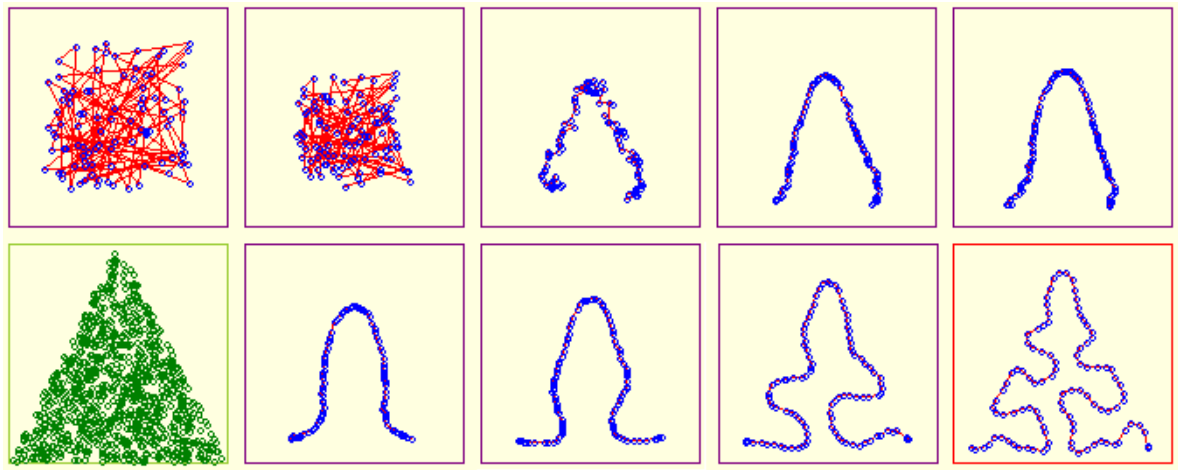


Fig. 10.46. Yet another example of mapping of two-dimensional input space into one dimensional neural topology

Take a look at figures 10.44, 10.45 and 10.46. It looks from them that one dimensional network pretty reasonably **fits in** a highlighted area of an input signal space – this guarantees two pretty important advantages.

- Firstly, a chain of neurons being a one dimensional network fills (better or worse) the whole selected area of an input signal space. It means that for every point of **two dimensional** input signal space there is a representative in a **one dimensional** neural network that will indicate its occurrence. There are no points or areas in the multidimensional input space that are „orphaned".

- Secondly, for the objects in an input signal space that lie close to each other (that is they are in some way similar one to another) – correspond adjacent neurons in the one dimensional chain of neurons. Unfortunately, **it is likely so, but it is not always**, so you should expect errors (fig. 10.47), but still in most cases, the fact that some two states of an input signal space are being represented by two adjacent neurons implies that they are similar states.

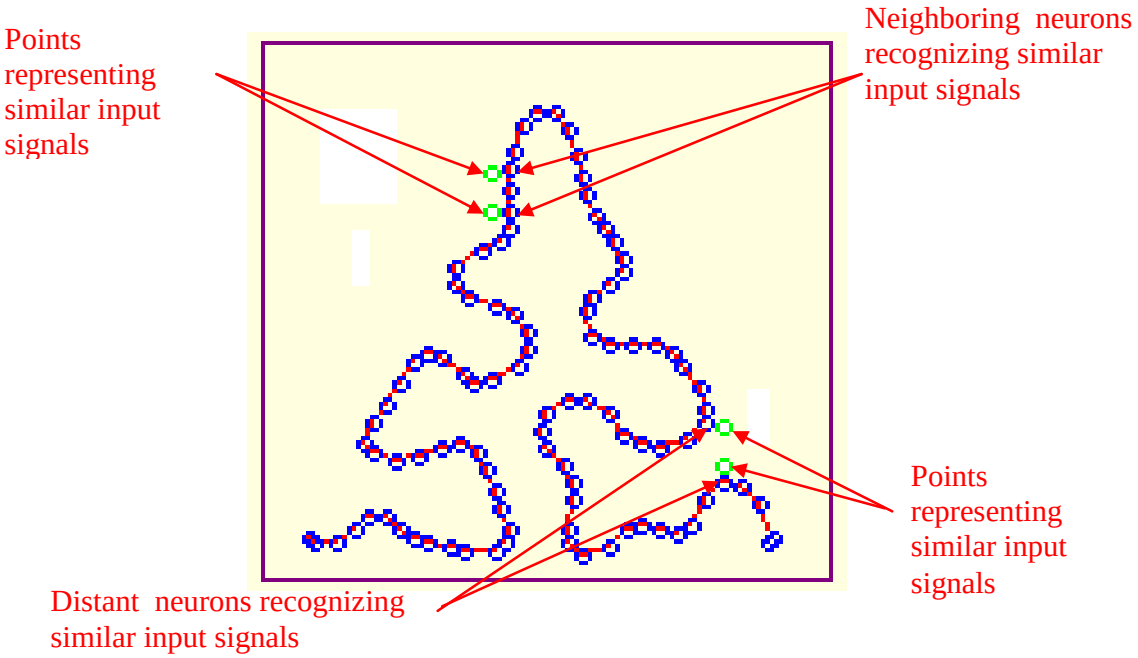


Fig. 10.47. Example of proper and non proper representation of similarity of input signal in neural network topology

Let's together consider what meaning may it have. Now in a great number of tasks related to informatics we encounter a similar and a very difficult problem. For getting the knowledge related to some phenomenon and some processes there are being gathered enormous quantities of data. E. g. to figure out the current state of a nuclear reactor in a big power plant it is necessary to measure and evaluate hundreds different parameters. The same refers to a blast furnace in a steelworks, a multi-engine aircraft during takeoff and also to a big company which we would like to effectively manage. An actual state of each of these big systems is described with hundreds and thousands of numerical parameters, we may imagine it then as a point in a space of great number of dimensions. This multidimensional space is necessary, because a result of each measurement, an effect of each observation, a state of each signaling device – should be put on a separate axis. As certain processes evolve, values of every parameter change and a point corresponding to a state of the considered

system changes its position. Therefore for full estimation of a plane safety, stability of a reactor, effectiveness of a blast furnace or a company profitability – all the time it is needed to evaluate position of a point in a multidimensional space. Specifically – the thing is about a position of such point that represents an actual state relative to areas for which we may assign a particular practical meaning: stable activity of a reactor – or a symptoms of overheating, good quality of produced pig iron – or its defective composition, favorable growth of a company – or a specter of bankruptcy.

In order to control and inspect a considered process we must have a current and certain information about its state, what is a trend of change and where it heads for. When trying to solve this kind of problems, in contemporary engineering, generally we use solutions where all original data is gathered and passed to a person which makes decisions. As an effect of that we can often find giant control rooms with many indicators and blinking lights, as the ones from nuclear power plants, that you may know from movies, or cabins of supersonic jets where every square centimeter is filled with some devices, meters and indicators, as well as kilometers long printouts containing many numbers and charts of economic rates on which businessmen rack their brains.

Such solutions though are fatally ineffective in practice. This mainly comes out of the fact that no man is capable of effective inspect, control and analyze of thousands of input data. In addition an operator of nuclear power plant, pilot of an aircraft or a chief executive of a company does not need such detailed data. He needs a synthetic global information: everything goes right or – there is happening something wrong. Such an information may be produced by a Kohonen Network.

Imagine that you have built a network where to its every neuron come few hundreds or even a few thousands of signals related to every collected measurement data. Such network is not necessarily more difficult to program than a network with two inputs, but it will require more space in computer's memory and more time during simulation of its working. Imagine that such network predicts two dimensional proximity of neurons, which you use so that an input signal of every neuron you will display in some (predetermined) point on a screen, and signals of neighbors you will display in neighboring rows and columns in order to visualize their relation. After teaching the network with using a Kohonen method you will obtain a tool making a specific “mapping” of multidimensional, difficult to evaluate, data, to a plane of one screen, which can be easily looked over, evaluated and interpreted. Image on a screen will provide data that can be interpreted as following:

- Every practically occurring combination of input signals will be represented by exactly one neuron (“winner”), which will detect and signal occurrence of this exact situation. If you will depict on a screen only an input signal of this particular neuron, you will obtain an image of a luminous point moving on a screen. If, based on previous studies, you will remember, which areas of screen correspond to good states of a supervised process and which to dangerous states – then by observing where actually the luminous point lies and where is it going – you can obtain a synthetic evaluation of situation from a perspective of the supervised process.

- On a screen you can also display output signals of neurons without making a discrimination following from usage of a principle of competition. You can make an agreement so that different values of output signals will be displayed in different colors and then a change in a supervised process will be displayed as a colorful mosaic. It carries much more information about a state of a supervised process and after obtaining some practice lets deep and accurate reasoning about observed trends, however at first sight is pretty illegible.

Ways for depicting results of Kohonen network working can be more. But yet, note on one common feature of them: they are always two dimensional images, convenient of being looked over with a glance and relatively easy to interpret. These images do not carry information about detailed values of particular data (they can be easily reached out when needed), but they show synthetic, overall image which is so very necessary to person that makes decisions and evaluates results. These images lower a detail level of the data being analyzed, but thanks to averaging and generalization described above, this way may be obtained very good results.

10.11. Control questions and self-study tasks

(Translation by Rafał Opiał, rafal.opial@op.pl)

1. What is the difference between self learning and self organization?

2. Kohonen Networks very often are called: tools that let „take a look into multidimensional spaces of data". Can you give reasons for such designation?

3. One of possible applications of Kohonen Networks is using them as a „novelty detector". A network in such application should signal a fact that here occurred a set of input signals that had never occurred before (neither in an identical form nor any similar). Automatic signaling of such situation may have essential meaning e.g. for detecting a theft of a credit card or a cell phone – thief usually use it in a different way than an owner. What do you think – in what way a Kohonen Network signalizes that it encountered a signal having character of „novelty"?

4. Study a course of a self organization process after a change of sub area of an input signal space from which come random input signals and which is filled with „grid" created by a network. Program lets you choose one of three shapes of a sub area: square, triangle or a cross. Analyze consequences of this choice. If you are advanced in programming, you may work a little on a program and add some another shapes from yourself.

5. Study what impact has a coefficient of learning of a neuron, named Alpha0 in the program. Changes of this coefficient cause acceleration of learning (when it is increased) or „calming" of a process when it is decreased. The coefficient is step by step automatically decreasing (see below), which effects in that a learning process – fast and dynamic at the beginning, becomes more stable as time elapses. You can also change this, and then analyze observed results.

6. Study what impact on a self organization process have changes of a coefficient of learning of neurons being neighbors of a winner neuron. The coefficient is called Alpha1 in the program. The higher the value of this coefficient the more visible the effect of „pulling" neighbors along the winner neuron. Analyze and describe results of changes of the coefficient value (as well as ratio Alpha0/Alpha1) and their impact on the network behavior.

7. Study what impact have different values of neighborhood range on a network's behavior and the self organization process. This number states how many neurons count into a neighborhood, i.e. how many neurons undergo forced teaching when a „winner neuron" is self learning. This number should depend on a network size, and exactly this is the way it is set by default in this program. However I suggest, as an exercise, careful examination of its impact on a network behavior. It is worth note that bigger numbers of neighborhood range visibly slow down the learning process.

8. Study what impact on a self organization process have changing of the **EpsAlpha** coefficient which controls the decreasing of coefficients **Alpha0** and **Alpha1** in every iteration. Note that the smaller

the coefficient is the faster will be decrease of learning coefficients and the faster the learning process will stabilize. You can – if you want – set a value 1 to this coefficient, then the learning coefficients will not decrease, or you can even set a value slightly bigger than 1 which will result in more „brutal” learning in every next step (see what will be happening!).

9. Study what impact on a network behavior and a learning process will have changing the range of a neighborhood. You can change coefficient **EpsNeighb** which controls the process of narrowing (in consecutive iterations) a range of neighborhood. Try comparing effects of its changing to effects you've observed regarding the **EpsAlpha** coefficient.

10. Study what effects cause attempts to "over teach" a network, which at the beginning learns to fill e.g. cross shaped subarea of an input signal space, then changed to rectangular or a triangular. Remember that in experiments regarding "over teaching", after a sub area change you need to increase values of **Alpha0** and **Alpha1** coefficients and a range of a neighborhood.

11. **Advanced exercise:** Modify the program so that it will be capable of modeling tasks where Kohonen Network deals with a phenomenon of highly irregular probability of occurrence of points coming from different regions of an input signal space. Conduct a self learning and notice that, in a taught network, much more neurons will specialize in recognizing signals coming from regions more often represented in a teaching set. Compare the effect with a structure of a map of a cerebral cortex presented in Fig. 10.40. What conclusions can you draw from this exercise?

12. **Advanced exorcise:** Write a program simulating behavior of a robot depicted in chapter 10.9. Conduct attempts to simulate gaining skills of associating the sensory stimulates describing a state of environment with behaviors favorable for a robot (causing beneficial changes in its environment). Study what kind of representation of a knowledge about a surroundings (simulated environment of a robot) will be achieved and used by a self organizing neural network being its „brain”. By changing the environmental conditions (which you stimulate, so you can impose any possible laws in it), study, which of the conditions robot can „discover” in its self organizing network and which turn out to be too difficult?

11. Recurrent networks

11.1. What is recurrent neural network?

(Translation: Artur Gorski, gorski.artur@onet.eu)

Owing to examples shown in previous chapters, you already know how the teaching of neural network is conducted – single or multilayer, linear or nonlinear, taught by supervisor or gaining knowledge on their own, briefly – almost free to choose.

Almost because you have not thought of recurrent networks yet, which include feedbacks. Feedback is a joint that recycles signals from further neurons to neurons from input layer or to previous hidden layers. (Fig. 11.1). Against all appearances it is really relevant and significant innovation. As you will see, network with feedback has essentially more upgradable abilities and computing capabilities then classical network, which permits only one-way signals flow – from input to output.

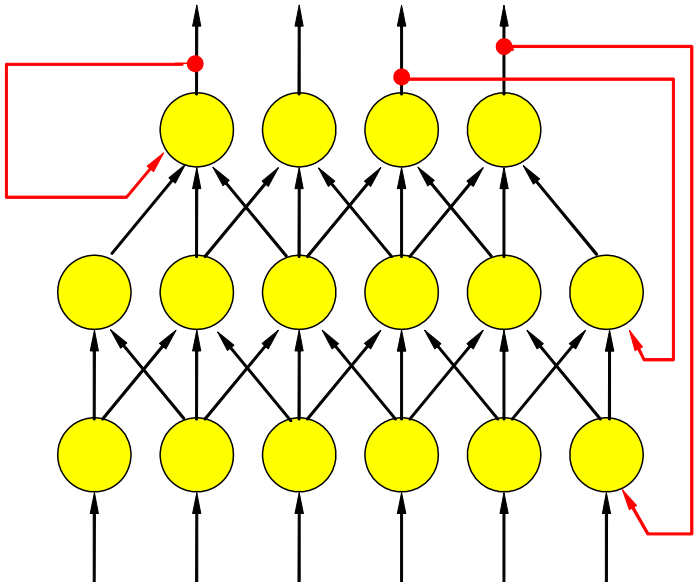


Fig. 11.1. Example of recurrent network structure. Feedback connections are distinguished by red color.

In networks with feedbacks will occur phenomena and processes, which you cannot find in one-way networks. Once stimulated network with feedback is able to generate thorough sequence of signals and phenomena, because signals from output (that are results of signal processing in certain 'n' iteration) while going back to neurons` input cause production of new signals, mostly completely different signals in 'n+1' iteration.

Specific phenomena and processes occur in recurrent networks because of complicated signals circulations – e.g. vibrations varying between alternate extremes rapid rise – equally rapid signals suppressing or mystifying chaotic roaming (which looks completely like undetermined progress).

Because recurrent networks with feedbacks are rather difficult to analysis, they are less popular. These difficulties come from the fact that signals can circulate through a network (from input to output than from output to input what again changes output etc.) therefore neural network responds to every input signal, even to signal given in short term, before all essential signals will be established, it goes through long sequences of various intermediate states.

Consider how it works using simple example. Imagine network which consist of only one neuron (linear for simplicity). It has two inputs – enter input signal by means of first one, than signal from output enter into the second input. Thus you will create a feedback (Fig. 11.2). It`s easy, isn`t it ?

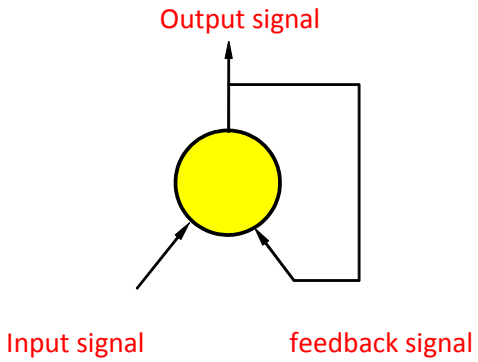


Fig. 11.2. The easiest network structure with feedback.

Now, check how this network works. For easiness **Example 12a**, in which you can determine network`s parameters on your own. Weight factor (feedback weight) by means of which signal from feedback will be entered into neuron, and also input signal (input signal strength), which runs network. In addition you can decide (by activating *single_input_impuls* or not), if input signal has to enter input continuously or temporarily (once), only at the beginning of simulation. This program will compute signals going around network, step by step, demonstrating network`s behavior (Fig. 11.3).

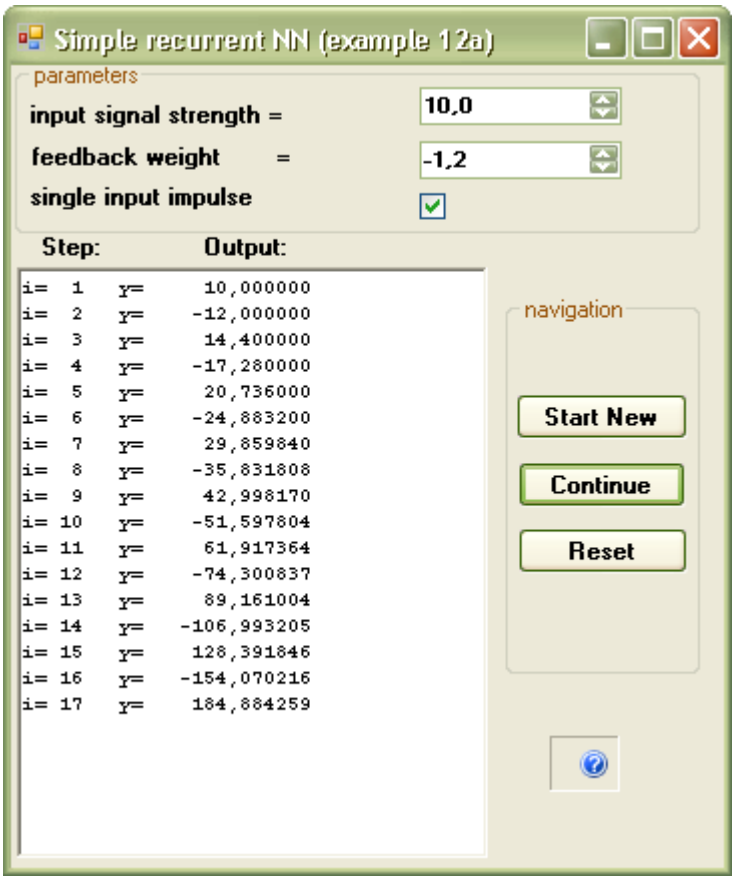


Fig. 11.3. This example shows complicated operation even with very simple input situation in system with feedback.

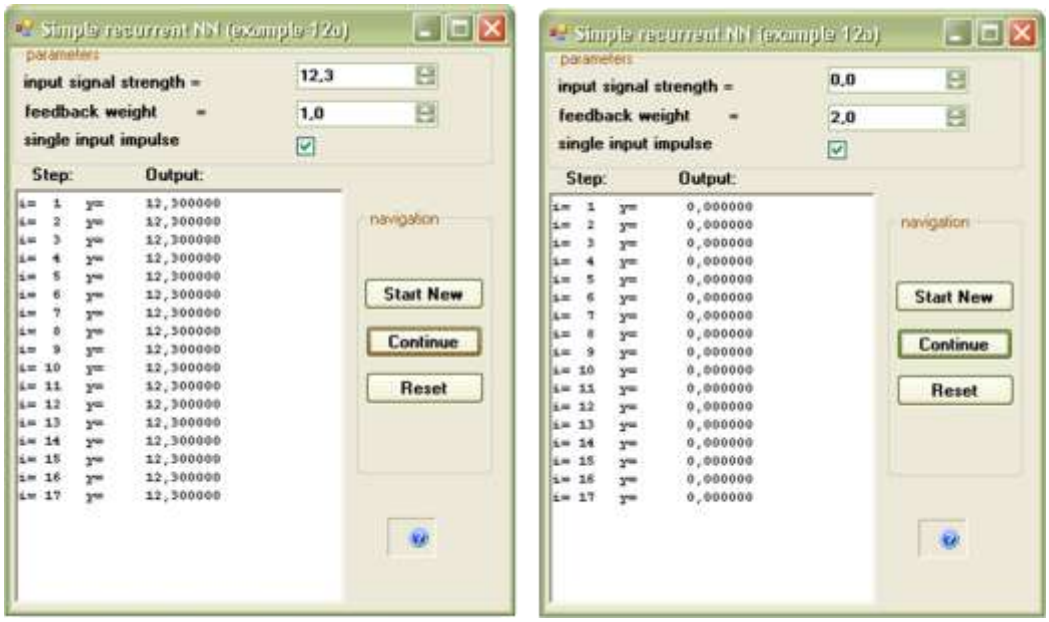
You`ll easily notice few regularities :

- This described network displays combined dynamics forms: after entering single (pulse) signal to the input – in the output sustains long-lasting process, during which output signal alters many times before it will reach state of equilibrium – as far as it`ll reach at all. (Fig. 11.3)

- Equilibrium in this simple network could be achieved (without output signal which lasts during whole simulation), only if a constant product of a certain output after multiplying by weight of a proper input, will give us output signal precisely equal with signal from feedback; that is needed to create output signal (signals at both neuron`s inputs will balance themselves out).

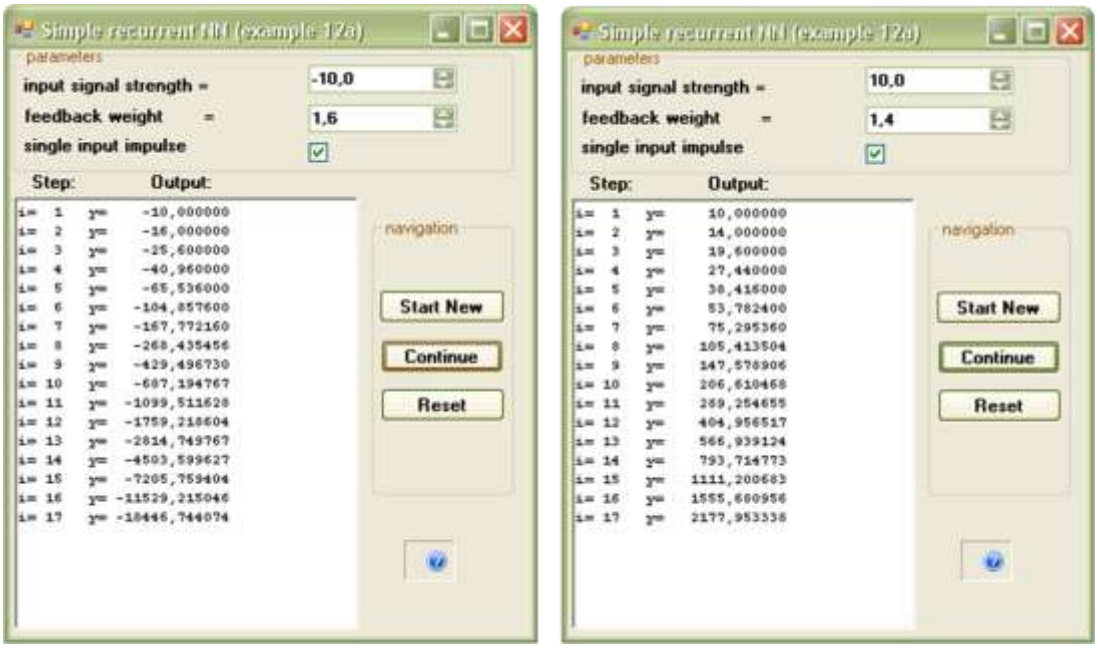
- Output signal, which complies with this condition, is called attractor. We will soon explain this definition thoroughly.

- Attractor location depends on network`s parameters. For network, which value of feedback weight factor is 1, every point is an attractor. While for any other network we can achieve state of equilibrium only when value of output signal is zero (Fig. 11.4). This feature is distinctive for this simple linear network. For nonlinear networks there could be more attractors and we will smartly make use of it.



11.4. Two ways of achieving state of equilibrium in linear network: zero signals (on the left) or single feedback amplification, without input signal simultaneously (on the right).

- If value of synaptic weight factor in feedback circuit is positive (it is called positive feedback) – progresses display aperiodic characteristic, in other words non-oscillating (Fig. 11.5). It is worth to notice that in this kind of the system, there could be process, which proceeds either through positive values (on the left side) of signals or negative (on the right side). While the time is passing, positive values are becoming more and more positive, the same happens to negative values. Whereas there are not any processes, in which signals would change from positive to negative and vice versa.



11.5 – Typical progresses in system with positive feedback.

- If value of synaptic weight factor in feedback circuit is negative (it is called negative feedback) – progresses display periodic character, in other words oscillating (Fig. 11.6).

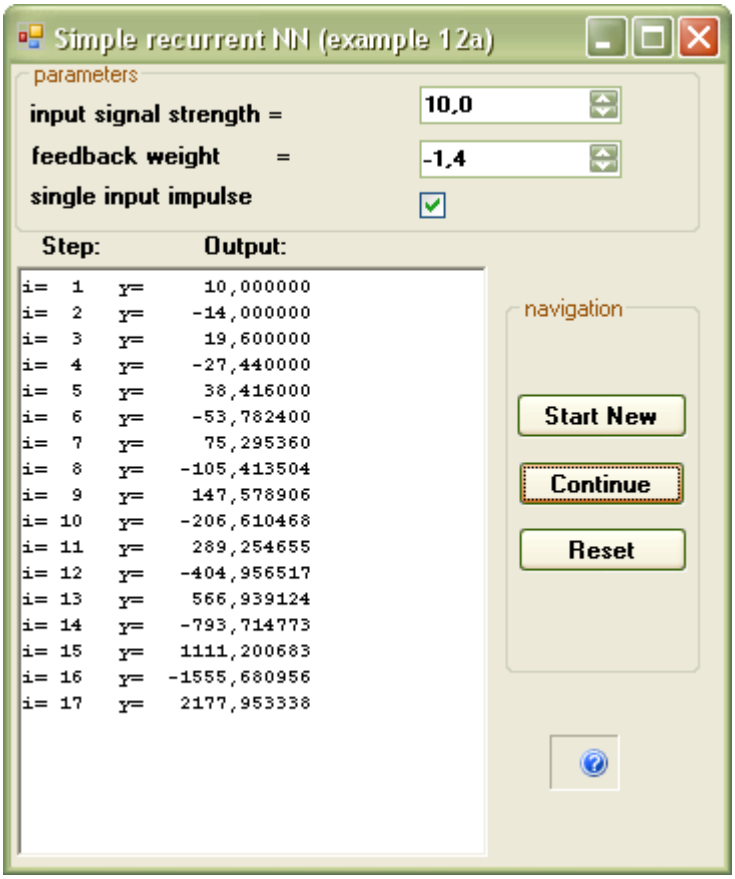


Fig. 11.6 – Typical progresses in system with negative feedback.

- In system with negative feedback, oscillating character depends on network`s parameters. Sometimes vibrations increase rapidly, network tosses between huge negative values and even more tremendous positive values and then generates bigger negative value etc. This network must result in catastrophe, because her behavior closely resembles behavior of human`s brain, which suffer from epilepsy or space rocket, which is taking off then is losing stabilization and finally is going to crush. However, vibrations sometimes lead up to zero, network is being stabilized – in this way work correctly conducted automatics systems.

- While observing behavior of systems with feedback you will see, that even small difference in parameters values will result in different (sometimes extremely) effects. Systems without feedback do not result in such differences, thus it`s recurrent neural network`s distinctive feature. (and other recurrent systems).

- Now let`s analyze described phenomena quantitatively. After few simulations you will notice, that if absolute value of weight factor in feedback is bigger than certain established value, called as stabilization point, then in system with negative feedback, as well as in system with positive feedback occur signals, which absolute values continuously increase (Fig. 11.5 and Fig. 11.6). These phenomena is known as unstable behavior.

- However, if absolute value of weight factor in feedback is smaller than established value – then circuit either with positive or negative feedback aim at state of equilibrium. (Fig. 11.7)

Fig. 11.7 – Stable progresses in system with feedback after decreasing value of weight factor highly below stabilization point.

Input signal could be also given for all the duration of simulation (just unmark checkbox *single_input_impulse* – Fig. 11.8). In this case state of equilibrium could be also achieved, but value of output signal (then network is stabilizing for this signal) is different and depends on value of input signal.

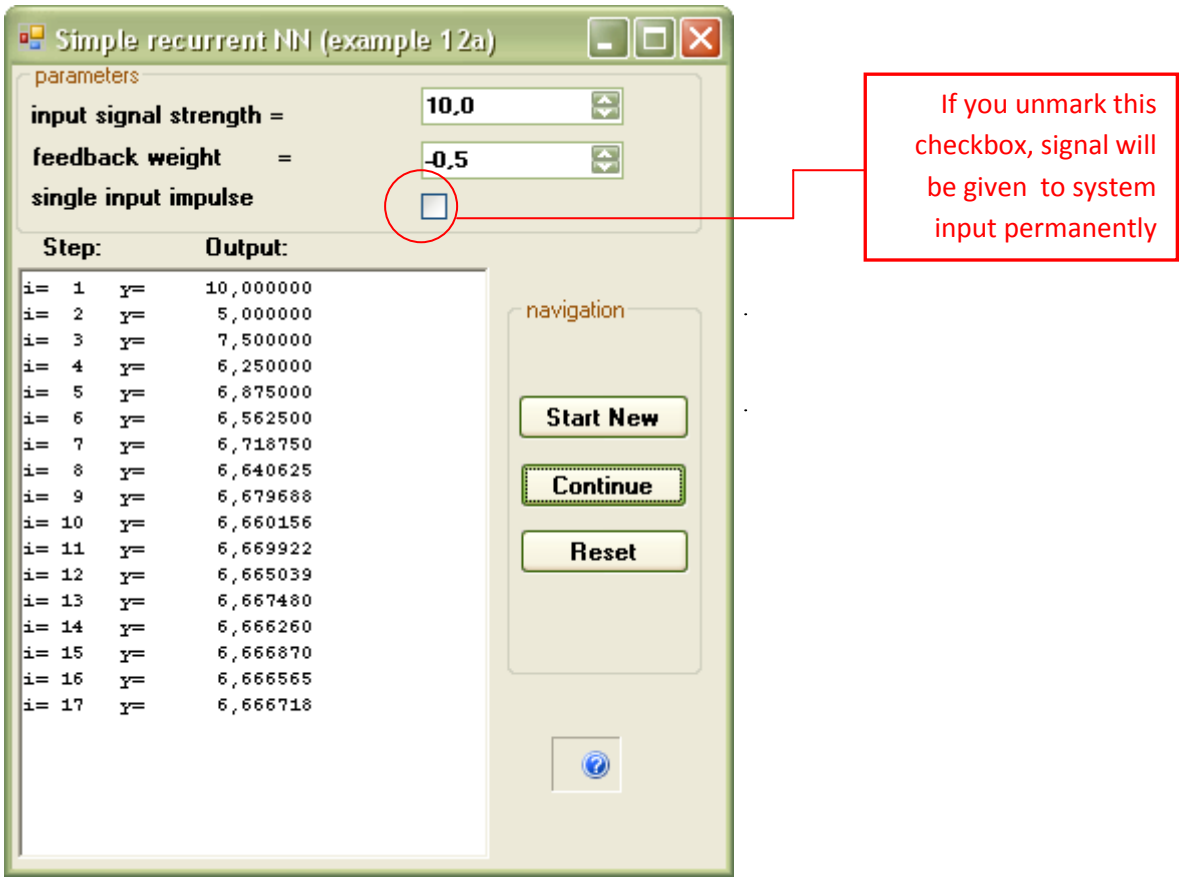


Fig. 11.8. Behavior of a system with feedback when established value of input signal is given permanently.

11.2. What features have network with feedback?

(Translation: Artur Górski; gorski.artur@onet.eu)

Let‘s summarize and draw conclusions from some conducted researchers, which you will find useful in your next steps.

Processing of input signals in network with feedback can display two different variability: if value of weight factor in feedback is positive (so-called positive feedback), signal changes aperiodically (in one-way), but when value of weight factor in feedback is negative (negative feedback, also called regulation), in network occurs oscillations (value of output signal alternates with smaller and bigger values, frequently with positive and negative. If neuron is nonlinear, system will be able to behave in a third way – chaotic signals roaming together with the greatest marvels of modern chaos theory (‘butterfly effect’, strange attractors, fractals, Mandelbrot‘s sets etc.) – but this is a different story. However, if you are willing to immerse yourself in these subjects – just Google it ! It‘s really worth !

Apart from dividing network behavior into two sections: aperiodical and periodical, you presumably noticed, that in networks with feedback, we can identify stable behaviors (signals have value restraints and usually after few iterations converge into certain established final value) and unstable behaviors – absolute values of successive output signals are bigger and bigger and finally they go

beyond acceptable values. Then in computer simulation will occur mathematical error (so-called floating-point overflow), but in electronic or mechanical systems something will very probably get burnt or explode!

Maybe it will be quite interesting for you, that in real neural networks these kind of phenomena happen every day, but probably you have not considered them in this point of view yet.

Positive feedback means that the more you think about something the more it is fascinating for you, so it takes you more and more time and finally it absorbs you completely. Surely, it has crossed your path many times, hasn't it ? And this is typical amplification effect of positive values of signals in system with positive feedback, just the same as one shown on the left side of Figure 11.5.

Similarly, you have also noticed inverse effect: if you have negative attitude towards something (e.g., school subjects, some entertainments etc.), then successive experiences confirm yourself in the belief, that it is foolish, not interesting and completely not for your taste. Point out that this situation is very similar to the process shown on the right side of Figure 11.5. If this closed circuit of positive feedback is not disconnected, it can result in psychic disturbances. It is especially dangerous when you strengthen in this way your attitude to someone. In case of amplifying negative it is called maniac psychosis, and when positive values are being amplified – in the 'head over heels in love' state and for no obvious reasons it is not treated. ☺

Note that in real nervous system also occurs (fortunately rarely) non-stability phenomenon. It is called epileptic seizure by neurologists, and disease in which occurs this phenomenon – epilepsy. Epilepsy is a grave illness, characterized by a brief disturbance of electrical activity in the brain (it is possible to detect them by means of registering electrical signal on cranium's surface caused by working brain, so-called EEG – monitoring), which activated result in rapid muscular contraction, they entail uncontrolled and vehement body's convulsions. In the past this illness inspired people with feeling of fear, or it was regarded as a sign of haunted either by evil spirit or good spirit ('*Saint Vitus Dance*'). Supposedly Julius Cesar suffered from this disease, so from tomorrow you could tell your friends that you simulate Julius Cesar's brain on your own computer, what should totally petrify them (factors of synaptic weight are equal zero).

I propose you to try by means of experiments implemented on your computer to think up or detect, what is the stabilization limit for this simple system with feedback. Try to discover on your own, which factors cause stable or unstable behavior. Because in next sentence I am going to comment it – I suggest you to hold on reading and experiment with program **Example12a**, and then compare what you have discovered with omniscient theory.

Undoubtedly while conducting researches with this program, you have noticed, that output signal alternations mainly depend on factors values; whereas input signal has weaker influence – even if signal will be given continually – not only in the first moment when program will start. While elaborating network behavior (or thinking over algorithm), you will easily see, that really important is whether absolute value of weight factor for feedback signal is bigger or smaller than 1. For factors smaller than 1 you permanently deal with stable process – aperiodic for positive values and oscillating for negative values. For factors bigger than 1 process is always unstable. While value of factor is precisely equal 1 and feedback is positive (positive feedback) you have a brush with strange

situation – all input signals turn out to be network attractor, and with negative feedback you deal with continual, neither rising nor suppressing oscillations. We call this state as stabilization limit.

In more complex networks, stabilization conditions are more complicated, so in case of calculating stabilization limit we have to use very advanced mathematical methods (Hurwitz's determinants, Nyquist's diagram, Lyapunow's theorem etc.) theory of neural networks with feedback has been for many years a great field for all kind of theorists, who zero on complex dynamics systems.

11.3. Who needs this kind of networks 'with loops' ?

(translation by Artur Górski gorski.artur@onet.eu)

Situation in network, which consists of only one neuron is cozily simple and easy to predict, so it relatively gives few occasions to observe interesting practical applications. That is why from program Example12a it is not possible to obtain any results that will associate with any useful effects. Only bigger networks with feedback, which include several or several dozens of neurons (especially nonlinear) could display really interesting and really complex behaviors, what is the key for their practical application. However, it is worth to know, that in this networks with bigger amount of neurons, which transfer output signals to each other by means of feedback, there is a possibility of more complex situations than stable and unstable behavior of simple single neuron, above described by mine conducted researches and (hopefully) also by yours. In this complex network, states of equilibrium could be achieved by different values of output signals (not necessarily this not interesting value 0), but it is possible to select certain joints structure and certain network parameters, that this achieved states of equilibrium will equal solution to certain problems.

It is the key for the most of nontrivial networks application. For instance, there are networks in which state of equilibrium is the solution of optimization problem (searching for the most beneficial decisions, which will ensure the highest profit or minimum loss by taking restraints into account – for example limited capital expenditure). In this group of networks, there are some, which by means of this method solve well-known traveling salesman problem (I have described it thoroughly in my book 'Neural Networks', which is available on the Internet), there were also conducted researches connected with optimal division of finite resources (e.g. water). My co-workers, students for the master's and doctor's degree made attempts (some very successful) to use networks for optimal structure selection, so-called portfolio selection while speculating on the stock exchange. Some researchers were also conducted personally. However, I am not going to write about it, because I would like you to improve your mind on getting to know secrets of neural networks functioning, and not endanger your wallet while speculating on stock exchange, where even with intelligent neural networks you will lost more often than win.

That is why in this chapter I am going to represent you another example, in which solution of a certain important and useful computer problem could be interpreted as achieving one among many states of equilibrium by network – so called associative memory.

This kind of memory has been a desire for most computer scientists for a long time, who are tired of current methods connected with searching for information in typical data bases, which are primitive. Let's look at this kind of memory and try to find it out how it works.

You definitely know that storing even millions records, which include some important information could be done effortlessly. But when it comes to find it very quickly, we could encounter some problems and delay. The task is simple when you know keyword, code word, specific value of record or anything else what differs this record from others. Computer will find it out and provide access to this information, in addition it could be done quickly and efficiently if base is indexed and constructed in an appropriate way. Undoubtedly you were doing it many times, while googling various enquiries and receiving answers (not always correct).

But it does not look so simple, when you do not know keywords or any other information identifying your request. However, you have general concept for which subject you need this information. Because you do not know how this desired information is exactly described – typical information search systems, which are included in databases or on the Internet, turn out to be unreliable and not effective. They pepper you with plenty of unnecessary information or they are just not able to find it out, in spite of the fact that your enquiry is available and could have been delivered. Excessive number of information is of course the lesser of two evils; among them there is required data, but how to dig them out? Most of you would describe it as a drudgery.

Though your brain works differently. Usually you need just a scrap of information to recollect anything you want effortlessly. Sometimes you need just one word, image, picture, conception, idea or formula and instantly you have complete set of information, references, suggestions and conclusions. Sometimes piece of fragrance, maybe even melody, sunset or specific place is enough to recollect series of memories, feelings and senses …

So based on a piece of information or by means of other information – somehow connected with this requested one, your memory is able to find it out. Our computers do not know how to do it, yet. What about neural networks?

Check it and you will see.

This task will be thoroughly described below, so you will become acquainted with details within few minutes. But before it happened, I have to tell you something. Associative memory issues are just a scrap of branch of knowledge so called cognitive science. It has awakened increasing absorption recently, especially among philosophers, pedagogues, psychologists, but also many specialists are passionately fond of this subject, especially physicists, who normally work with theoretical description of simple physical system (e.g. elementary particle) or specialized in describing system, composed of many particles, which interact each one (statistical thermodynamics). If you grub around library or the Internet, you will notice, that the number of physicists, who elaborated and set down lot of mathematically ambitious researches, is amazingly huge. These researches are strongly connected with behavior of network with feedback and processes, which lead to more beneficial direction of this behavior. Inter alia, that is how so called Boltzmann`s machines were developed (which operate by analogy to processes that occur in neural networks with thermodynamic phenomena, described by means of Boltzmann`s distribution) and algorithms of ‘simulated annealing’, but we will look at them next time. In USA professor Jacek Zurada from University of

Louisville and in Poland professor Leszek Rutkowski from Czestochowa University of mainly engage cognitive science and simulation of neural network. I have the honor to be a friend of this both highly talented scientists, I am willing to set down that they contributed the lion`s share to develop this theory, but '*amicus Plato, sed magnis amica veritas*' – thus I have to admit, that the biggest share to the theory of networks with feedback was contributed by American **John Hopfield**.

11.4. How Hopfield`s network is constructed?

(Translation: Artur Gorski, gorski.artur@onet.eu)

Undoubtedly, Hopfield`s networks are the most important and most often adapted subclass of recurrent neural network used in practice, that is why you should get to know them. If I had to generally typify them, I would claim that they are exact opposite to '*feedforward*' network, i.e. completely unpredictable with feedbacks, which were described in all earlier chapters. Networks in which feedbacks are allowed (so called recurrent networks), are able to contain certain number of feedbacks, however, an amount of them could be bigger or smaller. Whereas in Hopfield`s network feedbacks are just a principle. (Fig. 11.9)

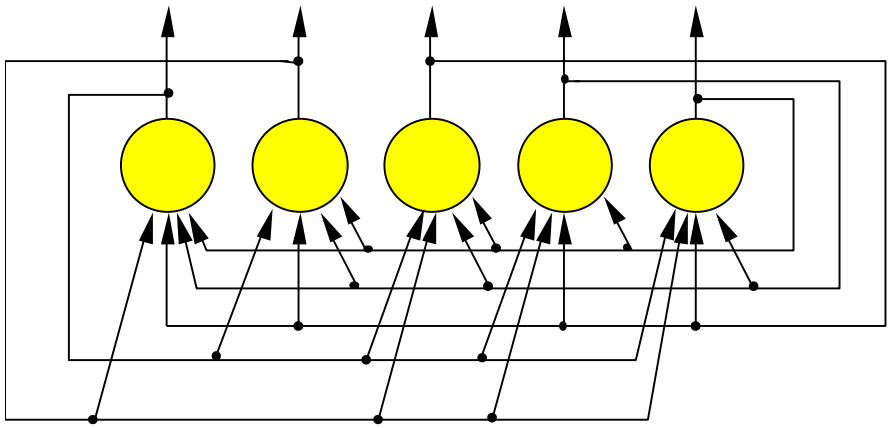


Fig. 11.9. Structure of simple Hopfield`s network

All joints, which occur in this network are feedbacks, all output signals are employed as inputs and all inputs transport feedback`s signals to all neurons. In Hopfield`s network each neuron is connected with all other neurons within entire network, this connection is based on two-sided feedback rule, thus network works and looks like a one feedback. Hopfield`s network is an exact opposite to all earlier described networks, which without exception were feedforward networks, and any feedback was forbidden.

Hopfield`s networks are so important because, all processes which occur in them are always stable, so they can be used safely as a tool to solve many various tasks without anxiety that everything could suddenly explode.

In Hopfield`s network processes` stableness was achieved by dint of application of three simple things:

- very regular (and easy to put into practice not only in the form of computer program, but also in the form of specialized electronic or optoelectronic circuits) internal structure of network was introduced, which based on the fact that within entire network neurons are

connected in the ‘everyone with everyone’ way. Try to recall that the same simple (but costly!) principle of shaping connections in network was also used in other earlier described networks e.g. network which was taught by backpropagation method; where connections between neurons from hidden layer and neurons from output layer were organized in the same ‘everyone with everyone’ way. Thus Hopfield`s network uses tested models, but there is one significant difference – connected neurons works not only as information transmitter, but also as information transceiver, and they define one collectivity (see Fig. 11.10), not separate layers.

- feedbacks that consist of one neuron are forbidden.

It means, that output signal cannot be given directly to his input, what as you can see is obeyed in network`s structure in the Figure 11.9. Notice that it does not rule out situation, in which output signal from certain neuron influences his own value in future, because it is possible to feedback by means of other ‘intercalary’ neurons, however, influence of this additional ‘intercalary’ neurons is definitely stabilizing;

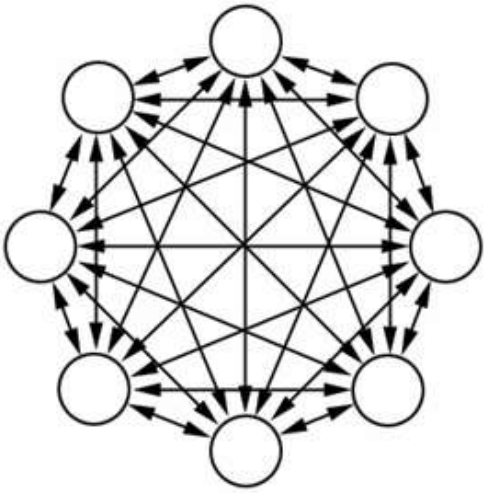


Fig. 11.10. Scheme of Hopfield`s network, which emphasizes equality of rights of each neuron and symmetry of connections.

- entered weight factors have to be symmetrical – it means that if connection from neuron ‘*x*’ to neuron ‘*y*’ is defined by certain weight factor ‘*w*’, then weight factor which defines connection from neuron ‘*y*’ to neuron ‘*x*’ has the same value – ‘*w*’ (Fig. 11.11).

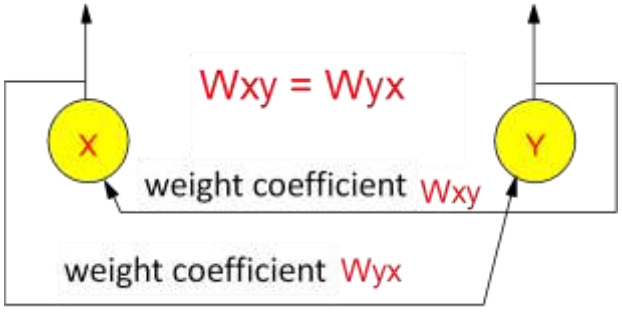


Fig. 11.11 – Obligatory rule of symmetry in Hopfield`s network.

We can fulfill all of these conditions effortlessly. The first two describe regular and very easy to achieve scheme of connections in network; if you choose Hebb`s method (described in chapter 9) to teach your network then the last one condition is automatically fulfilled.

Easiness of structure and application of Hopfield`s network results in very high popularity. They are numerously employed in various applications e.g. optimization tasks (described earlier) and also in generating certain sequences of signals, which follow one by one in certain (modifiable) rotation. By means of these networks it is possible to generate and forward control signals to various objects. Basing on this rule we control legs permuting in walking machines that stand on two, four or six legs (Fig. 11.12).

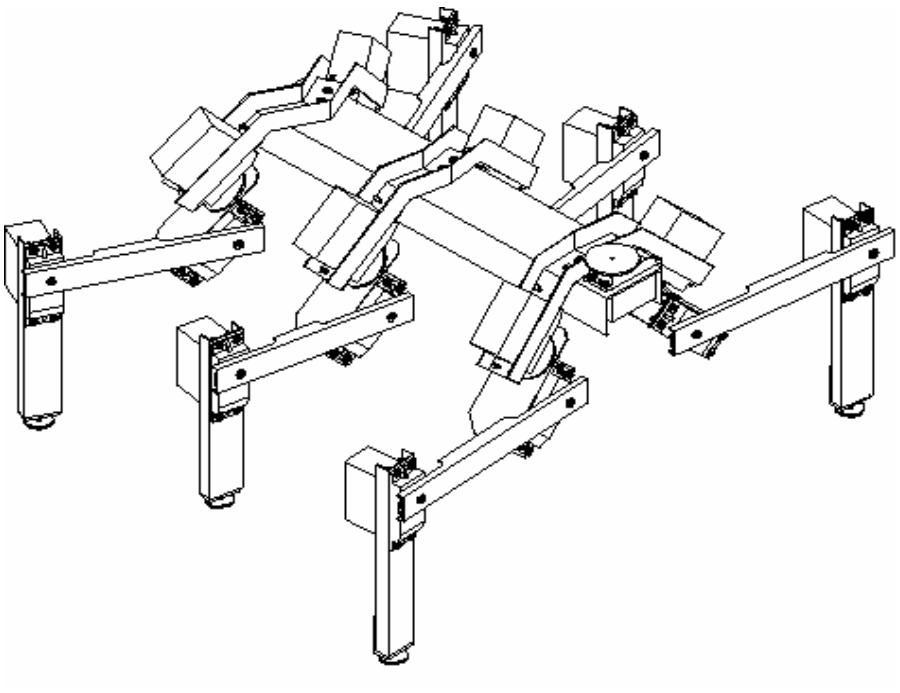


Fig. 11.12 – Example of treading robot, which legs are controlled by periodical signals – generated by 'brain' (Hopfield`s network).

'Brain' that control movements of this artificial machine always includes feedback network, then machine is able to periodically generate control signals on its own. Notice, that apart from number of legs, treading is always a periodical process, in which each leg is (in order) : lifted up, moved forward, lowered till the moment when the contact with surface is stable and moved back so as to transport 'body' of treading machine forward, afterwards, during some time, it performs a role of support, while driving force is provided by other limbs. Because treading on a rough surface requires not only periodical generating above described movements, but additionally mechanism, which will adjust machine for changeable situations (e.g. when one of the legs would get exposed in a crack), then as a 'brain' of a treading robot we have to use a tool that is able to: generate periodical behaviors, learn

and adapt for changeable situations. This conditions are thoroughly fulfilled by Hopfield`s network. So in most cases we use this kind of network or its simple modification.

These intelligently learning and treading robots are very interesting and will have in future huge application in exploring surfaces of remote planets, caves` interiors or oceans` floors, but for the time being we will skip it. In the following subsection I am going to show you, how we can make the best use of Hopfield`s network. In this case we will together construct associative memory.

11.5. How does the neural network work as an associative memory?

(Translation: Marcin Ptak, ornithion@gmail.com)

Program **Example12b** contains the Hopfield network model; its task will be to store and reproduce simple images. The whole "flavor" of this memory, however, lies in the fact that it is able to reproduce the message (image) on the basis of strongly distorted or disturbed input signal, so it works in a way that usually is called autoassociation[12]. Thanks to autoassociation Hopfield network can automatically fill out incomplete information. Figure 11.13 shows reprinted many times in various books and reproduced on the Internet (here downloaded from *www.cs.pomona.edu* website and also available on the *eduai.hacker.lt*) image showing how efficient the Hopfield network can be in removing interference from input signal and in complete data recovering in cases where there were provided only some excerpts.

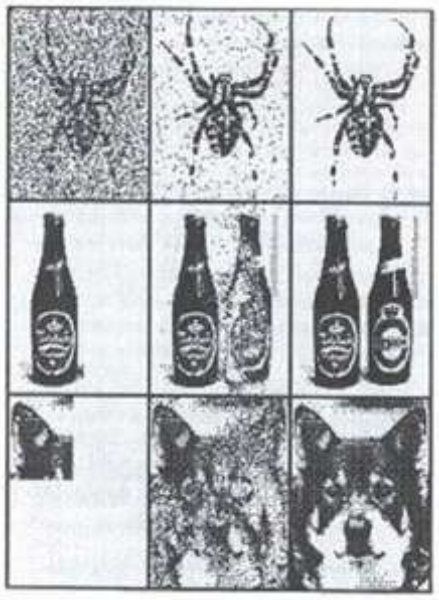


Fig. 11.13. Examples of Hopfield network working as associative memory. Description in text.

12 Autoassociation—means the working mode of the network, so that a specific message is associated with itself. Thanks to autoassociation application of even small fragments of stored information makes this information (for example, images) is reproduced in the memory in its entirety and with all the details. An alternative to autoassociation is heteroassociation, which consists in the fact that one message (for example, photographs of Grandma) brings memories of other information (for example, the taste of jam, which Grandma did). Hopfield network can work either as autoassociative memory or as heteroassociative, but the former one is simpler, and therefore we'll focus now on it.

The next three lines of this figure show each time on the left side a picture, which was provided as an input to the network, the middle column shows the transition state when the network searches its memory for the proper pattern, and - on the right side - the end result, that is the result of "reproducing" the proper pattern. Of course, before we could obtain here presented images of "remembering" by the network of relevant information (images) - first they had to be fixed in the learning process. During learning, the network was shown exemplary image of a spider, two bottles and a dog's "face" and the network has memorized the patterns and prepared to reproduce them. However, once the network has been trained - it could do really cool stuff. When showing her very "noisy" picture of a spider (the top row of Figure 11.13) - it reconstructed the image of the spider without noise. When shown the picture of a bottle - it remembered that during the learning process a picture of two bottles was presented. Finally, it was sufficient to show to the network the dog's ear only - to reconstitute the whole picture.

Pictures shown in Figure 11.13 are nice, but reproducing them is not associated with any important practical task. However autoassociative memory can be used for many useful and practical purposes. The network, for example, can reproduce a full silhouette of an incoming aircraft at a time when the camera recorded a picture of an incomplete machine due to the fact that the clouds obscured the full outline of the machine. This fact plays an important role in the defense systems, when the most crucial issue is the fast recognition of the problem, "friend - enemy". Network employed as an autoassociative memory can also fill out an incomplete inquiry addressed to some type of information system such as database. As it's commonly known, such database can provide many useful information, on the condition however, that a question is asked **correctly**. If the question to the network will be managed by untrained or careless user and do not correspond to preconceived models, then the database does not know how to answer it. Autoassociative memory can be helpful in "working things out" with the database management program, so finding the searched information will be done correctly even in case of imprecise (but clear) user's query. The network itself will be able to figure out all the missing details, that scatty (or untrained) user has not entered - although he should do that. Autoassociative Hopfield network then mediates between the user and the database, like a wise librarian in the school library, which is able to understand, what book the student asks for, wishing to lend a school lecture, and he has forgotten the author, the title and any other details, but he knows what this book is about.

Autoassociative network may be used in other numerous ways; in particular, it can remove noise and distortion from different signals - even when the level of "noisiness" of the input makes impossible practical use of any other methods of signal filtration. The extraordinary effectiveness of Hopfield networks in such cases comes from the fact that the network, in fact, reproduces the form of a signal (usually a picture) from its storage resources, and provided a distorted input image is only a starting point, "some idea on the trail of" the proper image—among all the images that may come into question.

But I think that's enough of this theory and it is time to go to practical exercises using the program **Example12b**. Due to clearness and readability, the program will show you the Hopfield network performance using the pictures, but remember that it is absolutely not the only possibility - these networks can just memorize and reproduce **any other information**, on the condition that we will arrange the way, the information is mapped and represented in the network.

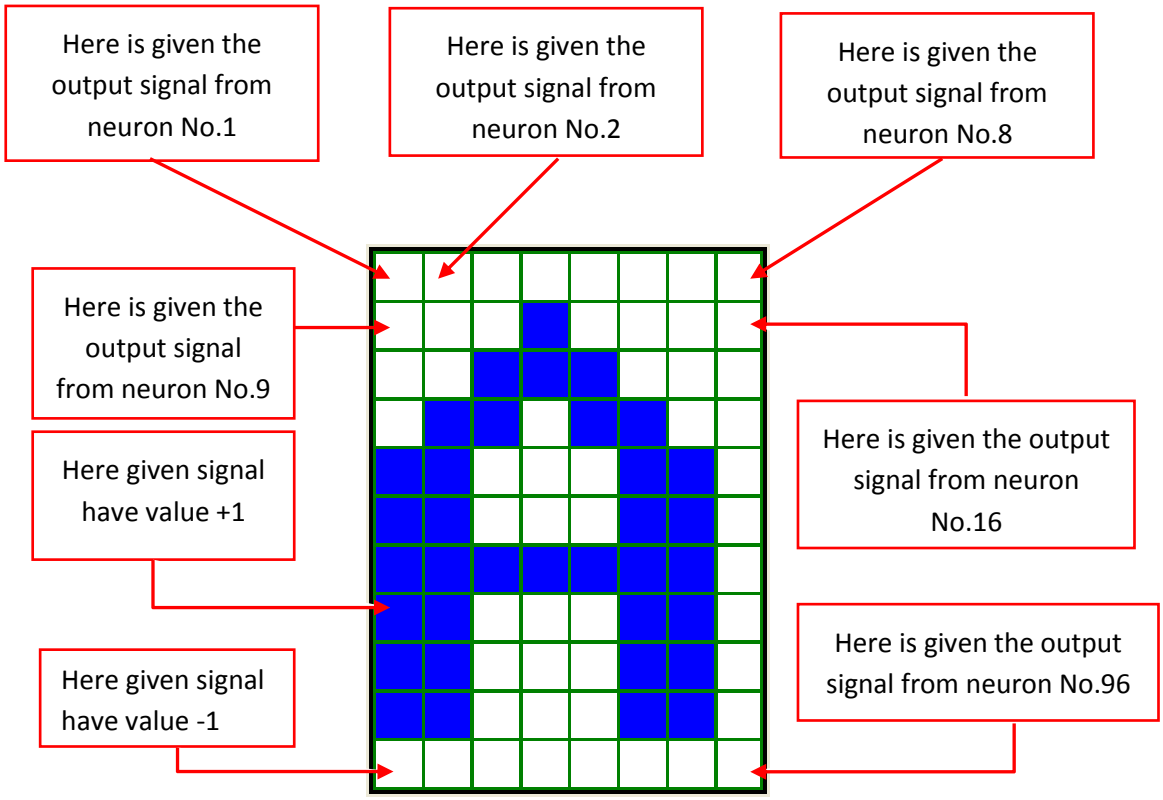


Fig. 11.14. This sample image presents the distribution of the output signals of neuronal networks.

So let us explain first the relationship between the modeled Hopfield network and the pictures presented by a program. We're connecting each neuron of the network with a single point (pixel) of an image. If the neuron output signal is **+1,** a corresponding pixel is black. If the output neuron signal is **- 1**, corresponding pixel is white. Other possibilities than +1 and - 1 we do not expect, because neurons the Hopfield network is build from, are highly nonlinear and can only be distinguished in these two states (+1 or - 1), but they cannot take any other values. Considered network contains 96 neurons that - only for the presentation of results - I put in order in the form of matrix of dimensions 12 rows and 8 items in each row. Therefore, each specific network state (understood as a set of output signals produced by the network) can be seen as a monochrome image with the dimensions 12 x 8 pixels, such as it is shown in Figure 11.14.

Considered pictures could be chosen entirely arbitrary, but for the convenience of creating a set of tasks for the network I decided that they will be images of letters (because it will be easy to write them using the keyboard), or completely abstract pictures produced by the program itself according to certain criteria in mathematics (I'll describe it more extensively further in text).

The program will remember some number of these images (provided by you or generated automatically) and then lists them itself (without your participation!) as patterns for later reproducing. In Figure 11.15 you can see, how such a ready - to - remember set of patterns may look like. Of course, a set of patterns memorized by the network may include any other letters or numbers that you can generate with the same keyboard, so a set given in Figure 11.15 should be considered as one of many possible examples.

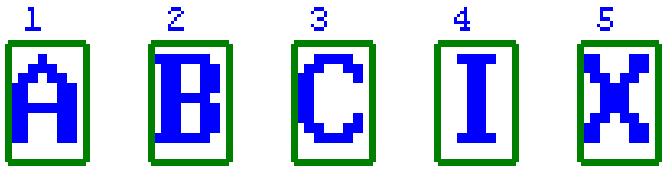


Fig. 11.15. A set of patterns prepared to memorize in the Hopfield network.

In a moment I will explain, how you can introduce more patterns to the program, but at the beginning I would like you to be focused on what this program is doing and what are the consequences - and the time for technical details will come shortly.

After the introduction (or generating) all the patterns, program *Example12b* sets the parameters (weight factors) of all neurons in the network, so that these images have become for the network points of equilibrium (attractors). What's the rule behind that process and how these setting of the weights are done - all this you can read in my book titled "Neural Networks". I cannot describe it at this time, because the theory of Hopfield network learning process is quite difficult and full of mathematics, and I promised you not to use such difficult mathematical considerations in this book. You do not need to know the details, after the introduction of new patterns into the memory; you simply click the button *Hebbian learning* shown in the Figure 11.16. This will automatically launch the learning process, after which the network will be able to recall patterns stored in it. As it's clear from the inscription on the button—the learning of this network is realized by Hebb's method, which you are already familiarized with, but at this time we won't be looking at the details of the learning process. For now just remember that during this learning process, there are produced the values of weights in the whole network, to be able to reach equilibrium state when on its output appear images matching to a stored pattern.

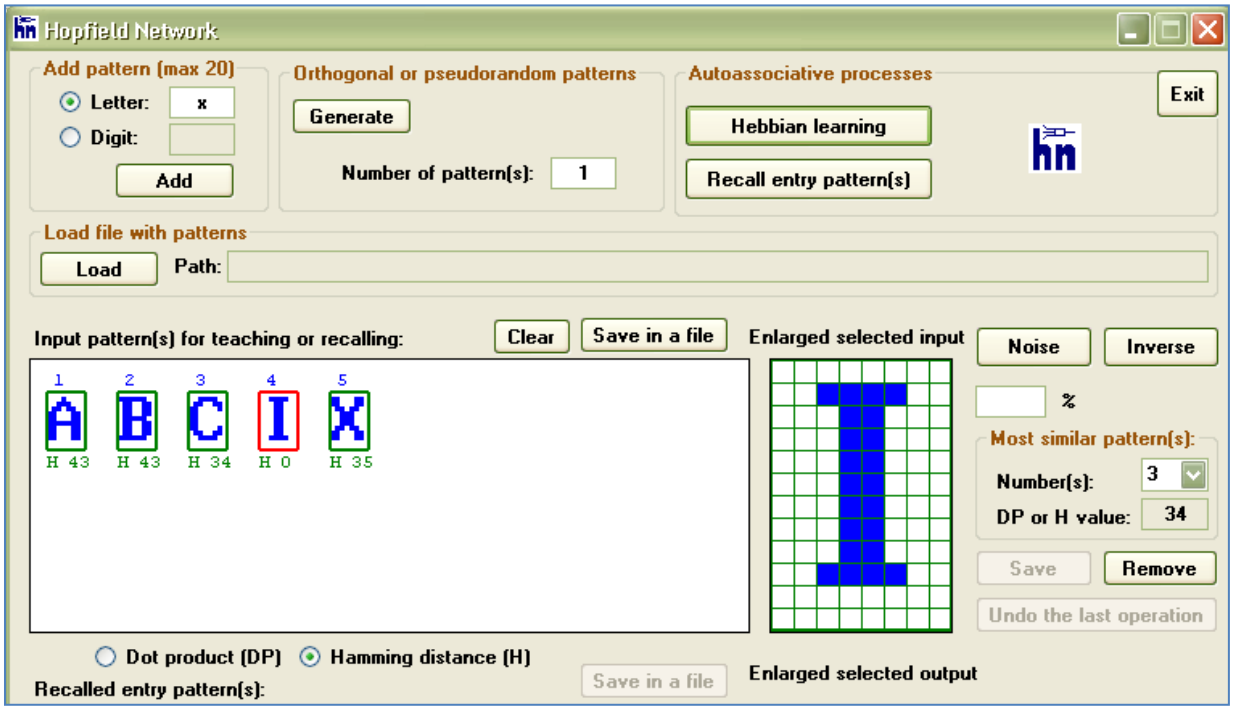


Fig. 11.16. Patterns to memorizing in network are entered into the **Add pattern.**

After learning process the network is ready to “test”. You can perform it yourself. For this purpose, first point (click the mouse) the pattern, the level of control you want to check for. Patterns and their numbers are currently visible in the **Input pattern(s) for teaching or recalling** window, so you can choose the one that the network has to remember. Selection (i.e. clicking the mouse) of any of these patterns will make its enlarged image appears in the window titled **Enlarged selected input**, and the chosen measure of its similarity to all other models will be visible directly under the miniature images of all the patterns in the window **Input pattern(s) for teaching or recalling** (see *Figure 11.16*). The level of similarity between images can be measured in two ways, that’s why beneath the window **Input pattern(s) for teaching or recalling** there are two boxes to choose from, described respectively: **DP** - the dot product, or **H** - the Hamming distance. What exactly does this mean I will describe you a little bit further, now just remember, that the **DP** measures the level of **similarity** between two pictures, so the high value of **DP** at any pattern indicates that this particular pattern can be easily confused with that one, you currently have selected. The Hamming distance is (as is also the name suggests), a measure of the **difference** between two images. So, if some pattern produces a high value of **H** measure, this pattern is safely distant from the one currently selected by you, while the low value of **H** measure indicates that this pattern could be confused with the current image of your choice.

Once you have selected the image that memorizing by the network you want to examine - then you can “torture” its pattern, because it is not difficult to recall a picture based on his ideal vision, but another thing is, when the picture is randomly distorted! Oh, that's when the network must demonstrate its associative skills - and that's what it’s all about.

Simply type in the box on the right, marked with the symbol **%**, the percentage of points the program is going to change in the pattern before it is shown to the network, so it can try to remind it. In the window marked **%** you can specify any number from 0 to 99 and that will be the percentage of points of the pattern the testing program will (randomly) change before it starts the test. But to enter this number into the pattern, you have to press the **Noise** button (i.e. noise or distortion). To make it even more difficult - you can enter additionally reversed (negative) image by pressing the **Inverse** button.

Selected and intentionally distorted pattern appears in the **Enlarged selected input** window. Look at it: would you be able to guess what pattern this picture was made from? Additionally, you can see if your modified pattern does not become after all these changes more similar to one of the 'competitive' images? Assessing the situation you will find useful measures of its “affinity” with different patterns, which will be presented in the form of numbers, located below the thumbnail images of all the patterns in the window **Input pattern(s) for teaching or recalling**.

I advise you: Do not set at the beginning to large deformations of the image, because it will be difficult to recognize from the massacred pattern its original shape - not just for the network, but also for you. From my experience I can suggest that the network is doing well with distortion not exceeding 10%. Good results are achieved when a - seemingly paradoxically - there is a very large number of changed points. This is the result of the fact that when large number of points is changed, the image retains its shape, but there is the exchange of point’s colors - white on black and vice versa. For example, if you choose the number 99 as a percentage of the points to change, the image changes into its perfectly accurate negative. Meanwhile, negative, is actually the same information,

which can be easily traced in the modeled network. When changing the number of points a bit less than 99% formed image is as well recognized for the network - it is a negative, with minor changes and the network also “remembers” a familiar image without any difficulties. However, very poor results are achieved when attempting to reproduce the original pattern with distortions ranging from 30% to 70% - network recalls something, but usually the pattern is reproduced after a long period of time (network needs many iterations before the image is fully recovered), and the reconstruction of the pattern occurs in a deficient way (still a lot of distortions).

So at the beginning of the exam you create an image of slightly (for example, look at Fig. 11.17, on the left) or strongly distorted pattern (Fig. 11.17, on the right), which becomes the starting point of the process of remembering the pattern by the network.

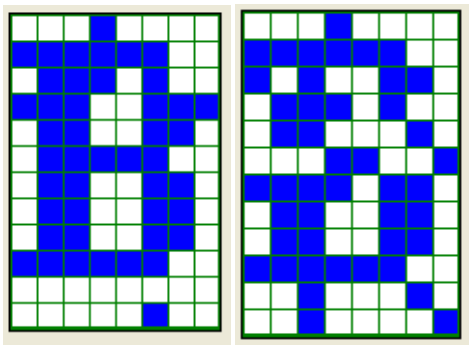


Fig. 11.17. Patterns, which the process of recalling messages stored on the network begins from: less distorted pattern of the letter “B” (on the left) and strongly distorted pattern of the letter “B” (on the right).

The process of remembering the memorized pattern is that the network output signals are given on the entry, and there (by the neurons) are transformed into new outputs, which, in turn, by feedback inputs are directed to a network, etc. This process is automatically stopped, when in one of the next iteration no longer occurs any change in the output signals (it means the network “remembered” the image). Usually this remembered picture is an image of a perfect pattern, which distorted version you applied the network - but, unfortunately, it not always goes this way.

If the image, which the process of “remembering” starts from, is only slightly different from the pattern - a recollection may occur almost immediately. This is illustrated in Figure 11.18, that shows, how network recalled the correct shape of the letter B, starting from a much distorted pattern shown in Figure 11.17 on the left. Subsequent images in Figure 11.18 (and in all further in this chapter) show sequentially (in order from the left to the right) sets of the output signals from the tested network, calculated and deducted from the model. Figure 11.18 shows that the output of the network has reached the desired state (the ideal reproduction of the pattern) after only one iteration.

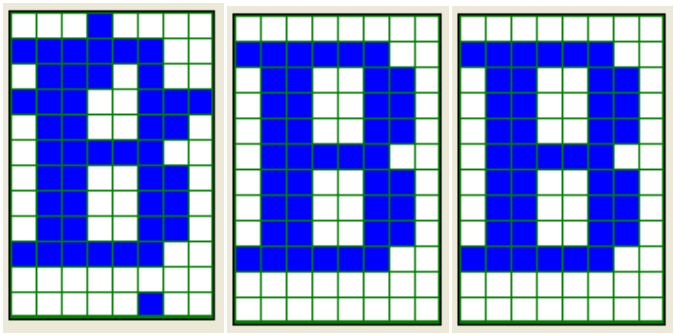


Fig. 11.18. Fast reproduction of distorted pattern in Hopfield network working as associative memory.

Slightly more complex processes were merged in the tested network while reproducing a highly distorted pattern, shown in Figure 11.17 on the right. In this case the network needed two iterations to achieve success (Figure 11.19).

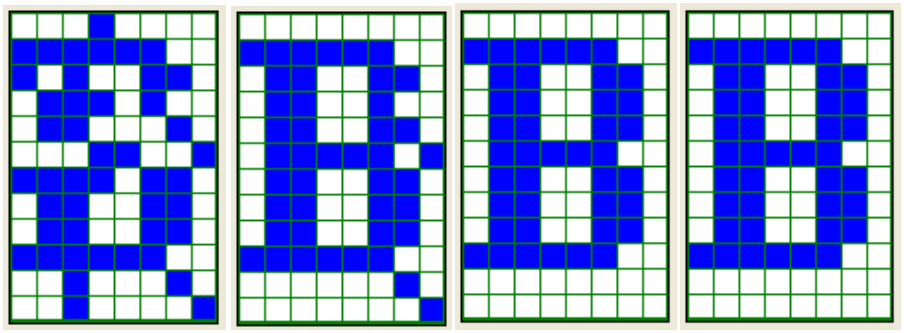


Fig. 11.19. Reproduction of highly distorted pattern in Hopfield network.

One of the interesting characteristics of the Hopfield network, you will learn when you will do yourself some experiments with the program, is its ability to store both the original signals of successive patterns, as well as signals that are the **negatives** of stored patterns. It can be proved mathematically that this is happening **always**. Each stage in the learning process of the network, which leads to memorizing a pattern, automatically causes the creation of an attractor corresponding to negative of the same pattern. Therefore, the pattern recovery process can be completed either by finding the original signal - or finding its negative. Since the negative contains exactly the same information as the original signal (the only difference is that in places where the original signal has a value 1, in the negative is - 1, and vice versa), therefore, finding by the network a negative of distorted pattern is considered also as a success. For example, in Figure 11.20, you can see the process of reproducing a heavily distorted pattern of the letter **B**, which has ended with finding a negative of this letter.

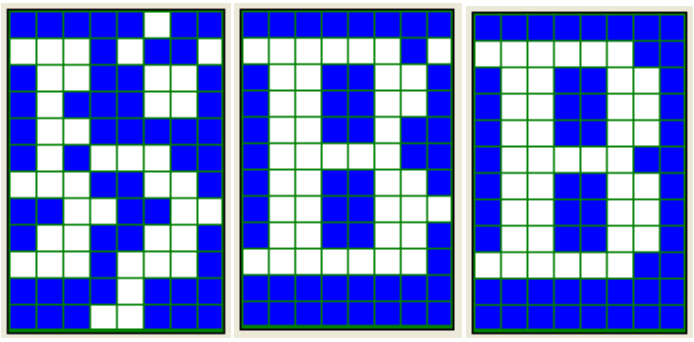


Fig. 11.20. Associative memory reproducing the negative of remembered pattern.

11.6. How works program for investigating Hopfield network by yourself discovery?

Not translated yet …

11.7. A few interesting examples

Not translated yet …

11.8. How and why we can use automatic patterns generation Hopfield network?

Not translated yet …

11.9. What studies can be performed on the associative memory?

(translation by Beata Słomińska; slominska.beata@gmail.com)

With use of the program **Example12b** you can conduct a series of studies, which will help you to understand the nature of associative memory constructed with the Hopfield network. I have shown you already how to inscribe information into the memory and how is the information reproduced. Now we will try to describe the capacity of the associative memory constructed with use of the neural networks.

In a standard RAM or ROM memory installed in your computer, capacity of the memory is restricted. This is due to the fact that every single information in this memory is inscribed in different location. You acquire access to certain location by typing – directly or indirectly – its address. The same follows for hard disk and CD. This implies that amount of information, which you can inscribe in every type of memory presented above is precisely defined by an amount of the addressed locations designed for storage. That is why an answer to a question on the amount of space in the memory is given immediately and precisely.

A memory constructed in the Hopfield network follows a different pattern. There are no separate locations designed for storage of particular information, as in the process of memorization – the same applies to recall – **all neurons** are engaged in learning of a given pattern. This implies that in particular neurons the patterns must overlap, what in turn causes problems (do you remember the "overhearing"?). What is more, a method of information recall (reading) from the neural memory differs from the one applied in computers. Instead of indicating name of a particular variable which contains certain information or indicating file name (both methods refer to the **addresses** of a message; their synonyms are the names of variables or files) – in the associative memory you are giving information as such – even incomplete or distorted. Thus, it is extremely hard to examine what is the upper limit of messages that you can introduce into the neural network.

Obviously, we dispose a precise theory, which explains all these issues in detail. If you ever needed this theory, you may find it in my book *Neural networks*, which I have cited here repeatedly.

However the aim of this book is to introduce elementary ideas and concepts on the neural networks by experimenting with them. By conducting experiments with networks we will try to answer the question on capacity of the neural networks. This path will be appropriate for us to deliver you the most precise information on the neural networks; but this information you will discover yourself by experimenting, what in result will give you truly understanding of the neural networks.

First and the most important factor deciding on simplicity and reliability of recalling the memorized signals is the amount of data recorded in the network; it takes a form of memory traces. After memorizing relatively small amount of patterns, for example 3 patterns, recall of information is reliable even in case of relatively significant distortions (Fig. 11.43).

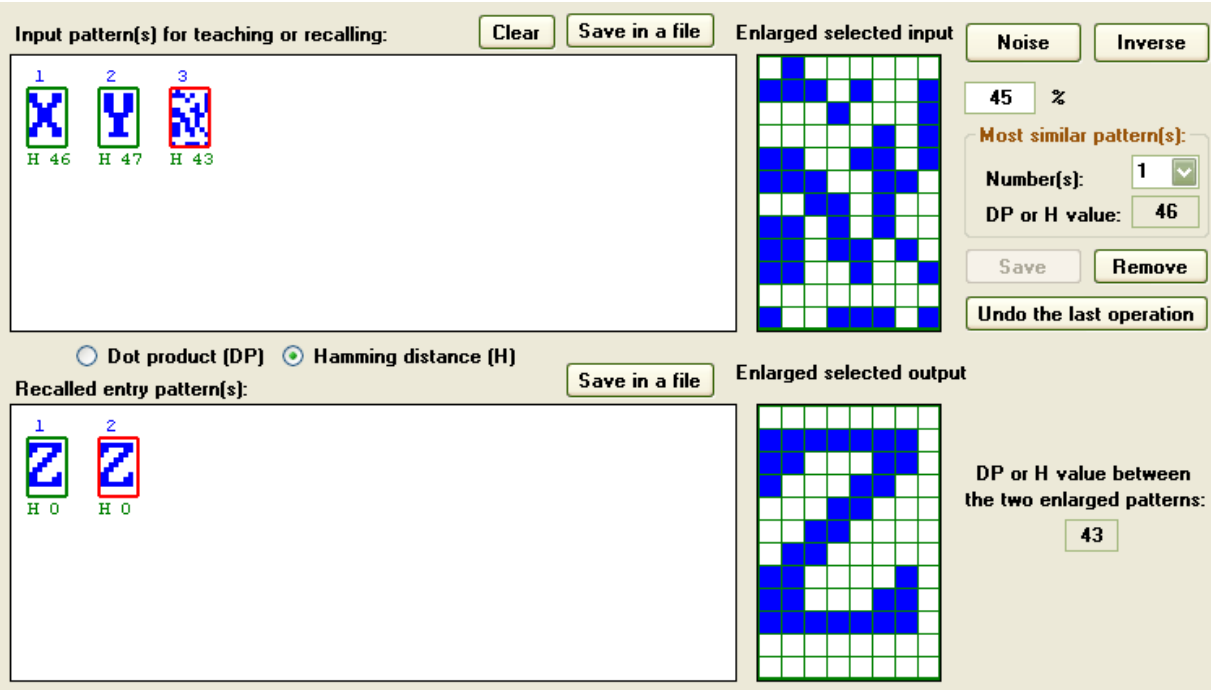


Fig. 11.43. Reliable recall of information for small amount of patterns

However the effect of recall depends also upon the difference between memorized information. On the figure 11.44 you will find results of an experiment conducted on a network with small number of

patterns but with high degree of their similarity. The latter is the reason of improper recall of the patterns.

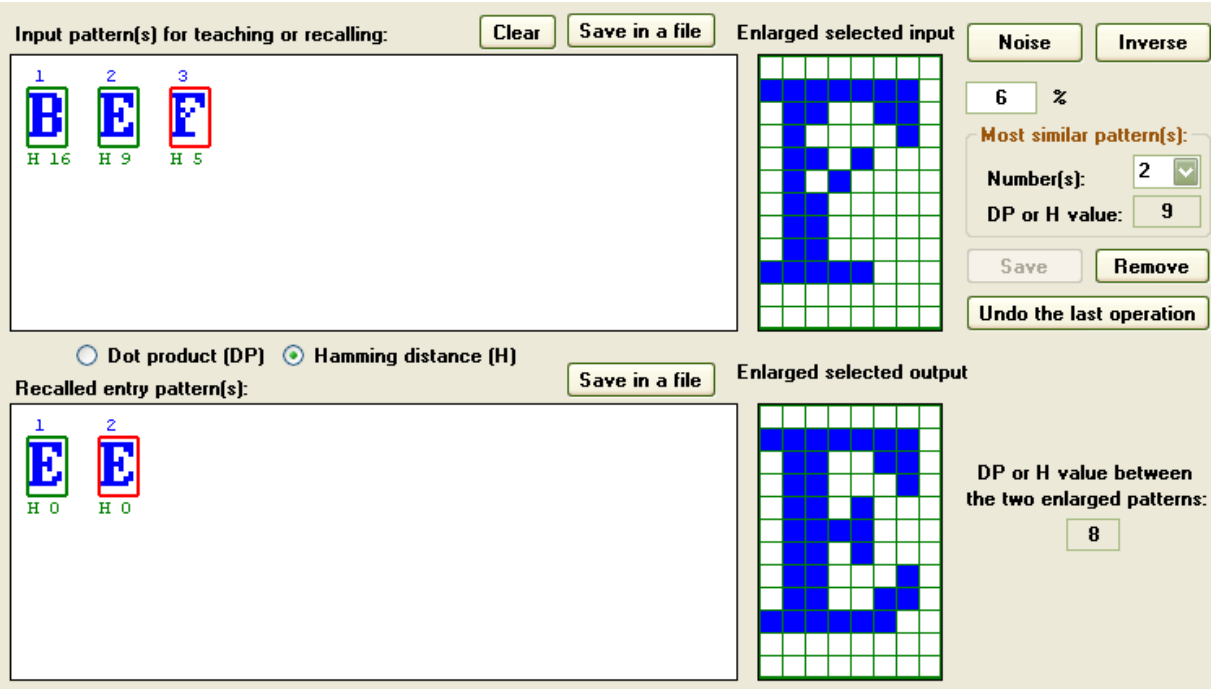


Fig. 11.44. Failure observed when number of number of patterns is rather small and patterns are very similar

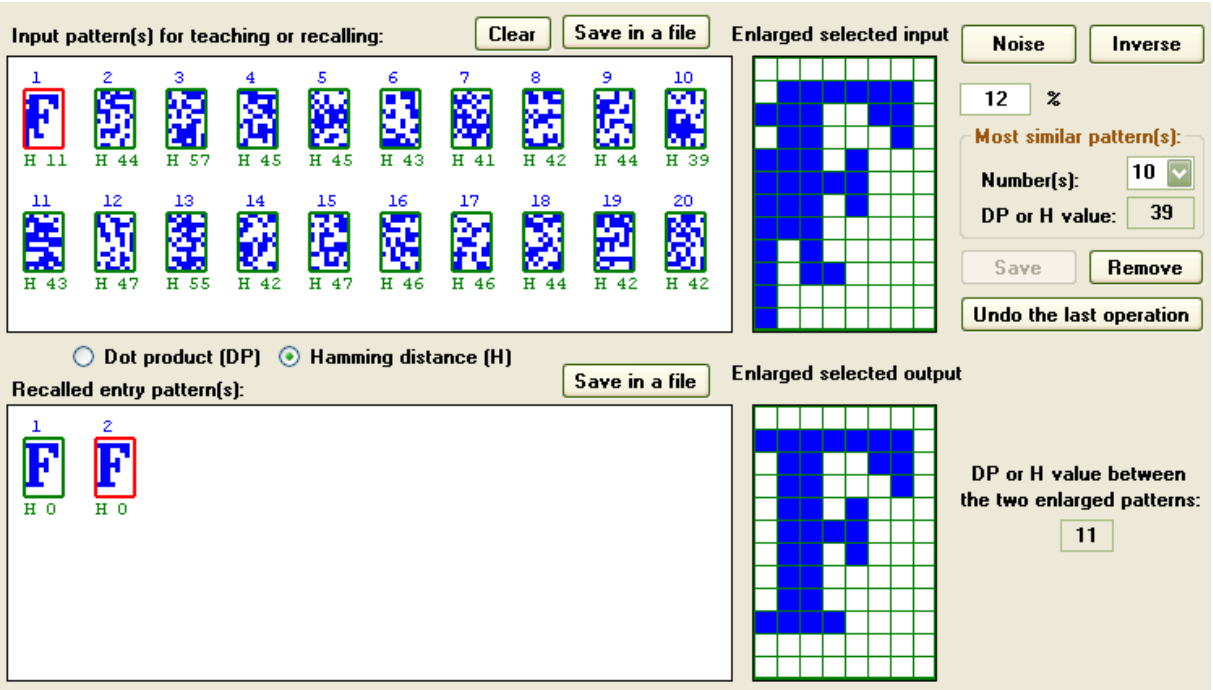


Fig. 11.45. Proper reconstruction of very destructed input signal when maximal number of pseudorandom patterns are used

Therefore if you intend to examine the maximum extent of network capacity you should operate with highly differentiated signals, mostly desirable with orthogonal or pseudorandom signals. Only

then you can achieve good results even with maximal number of patterns, that is in case of this program 20 (Fig. 11.45).

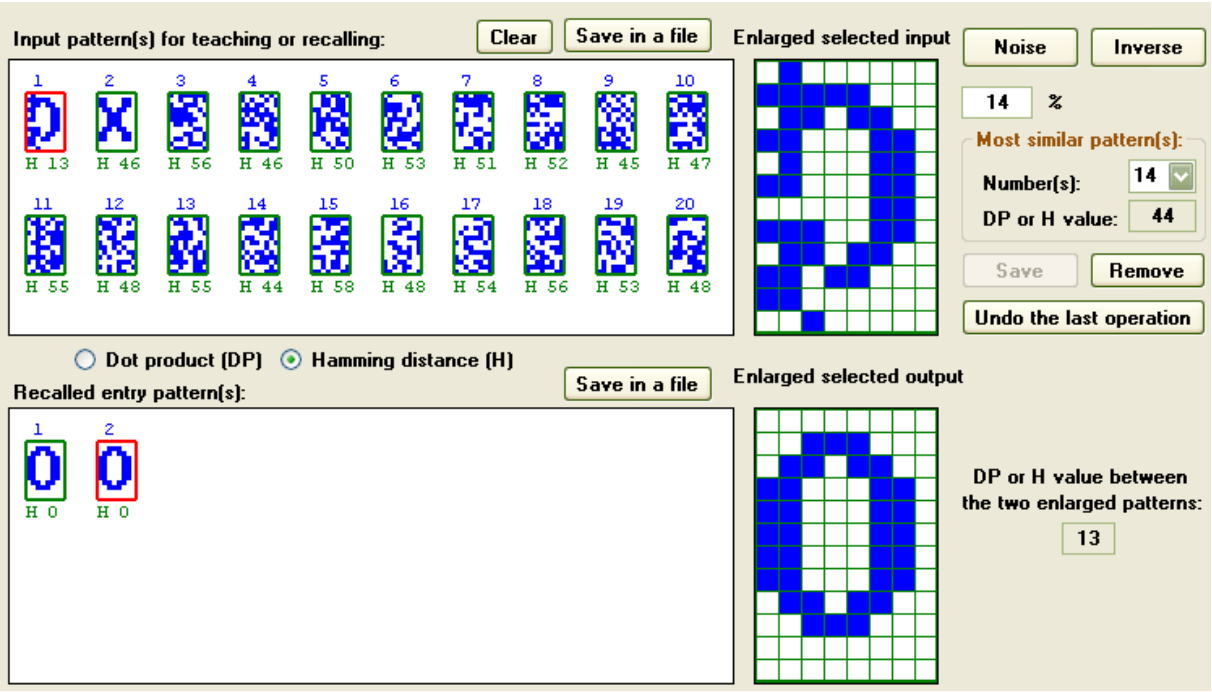


Fig. 11.46. Success obtained when big number of non exactly orthogonal patters are remembered

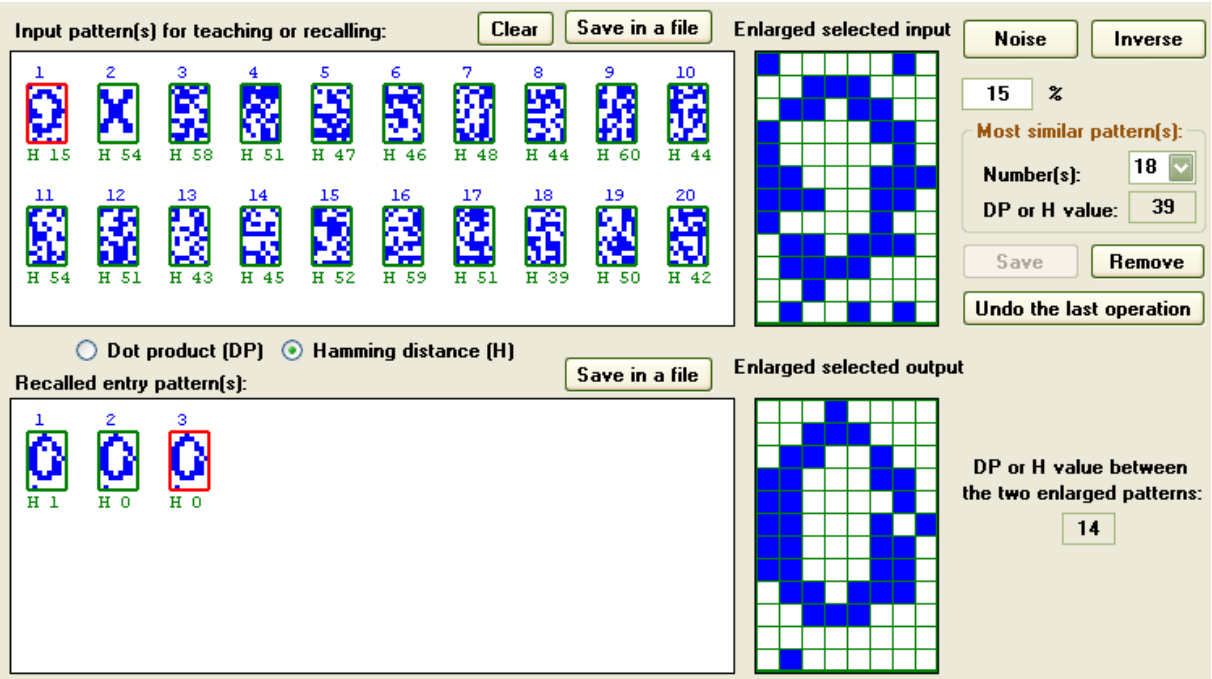


Fig. 11.47. Overhearing appearing when input signals are have destroyed by noises

With non-perfect orthogonal patterns, for example pseudorandom patterns, which emerge if you start running a process of automatic generation after giving two signals selected by you, results may appear still pretty good. It will happen only if you select highly different initial signals and with

relatively small distortions of initial images (image 11.46). Application of more distorted entrance in this case will expose existence of strong overhearing in the networks (image 11.47).

Proper recall of patterns in a system with generated by the program orthogonal or nearly orthogonal signals may be covered up by the fact that in case of a large number of memorized signals difficulties in recalling the entrance images are far greater than in case of low number of memorized signals. Even though you may notice that on the image 11.48 where slightly higher number of patterns is non-orthogonal, even with the rest of patterns being orthogonal, the network produces truly nasty overhearing; it happens even in case of insignificant deformations of the entrance signal.

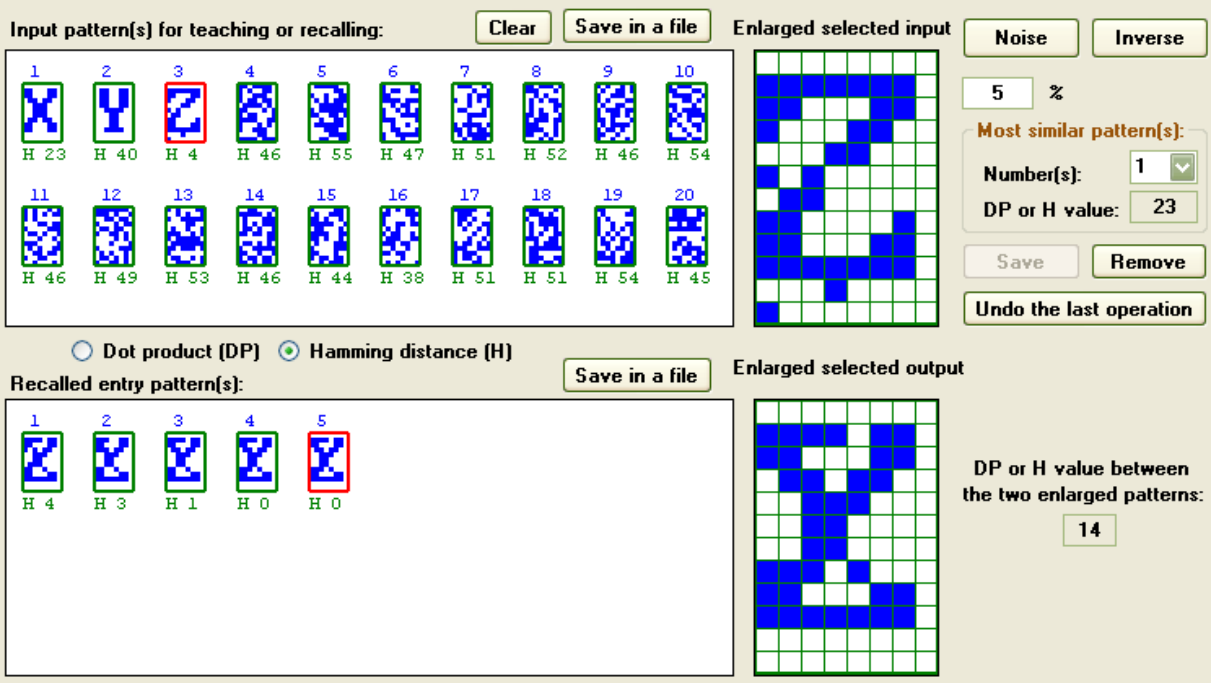


Fig. 11.48. Big overhearing observed also for minimal destroyed input signal when remembered patterns are not orthogonal.

11.10. What more it is worth observe in associative memory?

Not translated yet …

11.11. Control questions and self-study tasks

Not translated yet …