ARTIFICIAL AND COMPUTATIONAL INTELLIGENCE

Classic, Deep, Wide, Broad and Cascade Artificial Neural Networks of the 1st and 2nd Generations

Adrian Horzyk

horzyk@agh.edu.pl

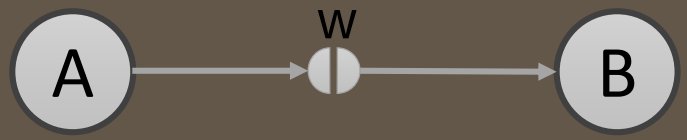
AGH

AGH University of Science and Technology Krakow, Poland

Brains and Neurons

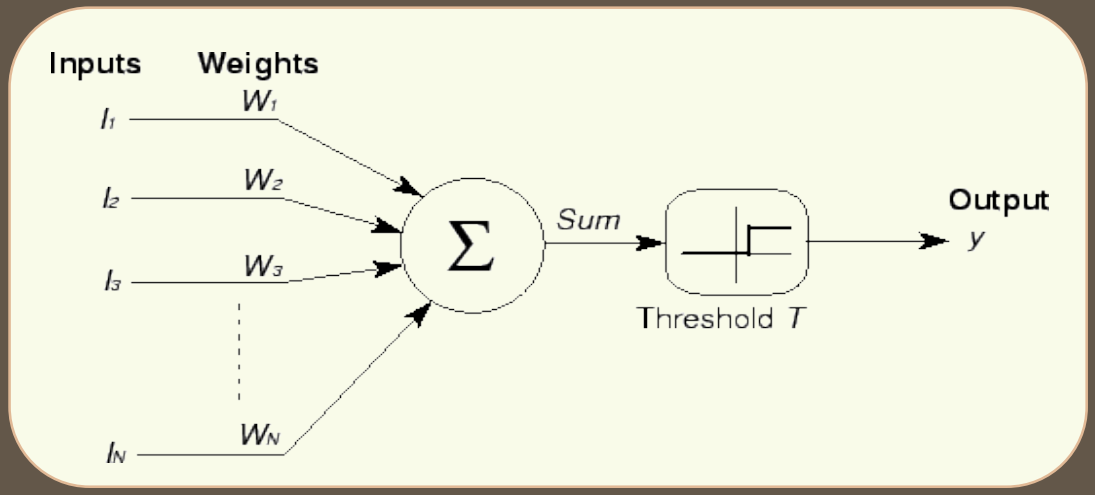
Brains and real neurons inspired scientists to develop different models of artificial neurons and their networks.

Hebbian Learning Principle (the first one) states that „*when an axon of [neuronal] cell A is near enough to excite a [neuronal] cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency [w], as one of the cells firing B, B is increased*". [D. O. Hebb, 1949.]



This principle assumes that a connection between neuronal cells is weighted, and the weight value [w] is a function of the number of times of presynaptic neuronal firing that passes through this connection, which takes part in firing the postsynaptic neuron.

The McCulloch-Pitts model of neurons (1st generation) implements only the most fundamental mechanism of weighted input stimuli integration and threshold activation function leaving aside issues of time, plasticity and other factors.



Hebb's and Oja's Learning Rule

Hebb's learning rule defines the weight of the connections from neuron j to neuron i:

$$w_{ij} = x_i \cdot x_j$$

where x_i and x_j are the inputs equal to 0 or 1 for neurons i and j, where i≠j, updated after each presentation of the training pattern, or after presentation of all (p) patterns:

$$w_{ij} = \frac{1}{p}\sum_{k=1}^{p} x_i^k \cdot x_j^k$$

where x_i^k is the k-th input of the i-th neuron.

Generalized Hebb's learning rule is defined for the postsynaptic response y_n:

$$\Delta \boldsymbol{w} = \boldsymbol{w_{n+1}} - \boldsymbol{w_n} = \eta \cdot \boldsymbol{x_n} \cdot y_n$$

Oja's learning rule is a single-neuron special case of the generalized Hebbian algorithm that is demonstrably stable, unlike Hebb's rule.

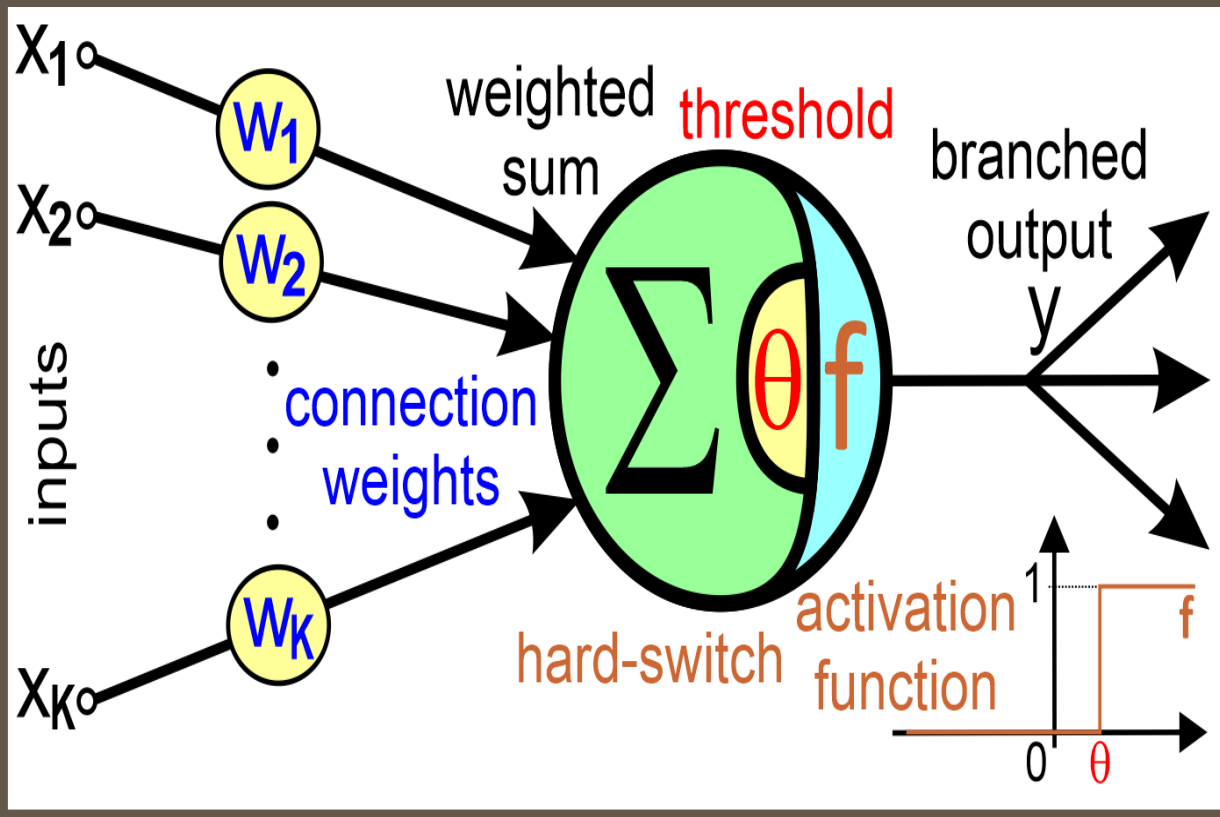
The change in presynaptic weights w for the given output response y_n of a neuron to its input x_n is:

$$\Delta \boldsymbol{w} = \boldsymbol{w_{n+1}} - \boldsymbol{w_n} = \eta \cdot \boldsymbol{w_n}(\boldsymbol{x_n} - y_n \cdot \boldsymbol{w_n})$$

where η is a learning rate which can change over time, and n defines a discrete time iteration.

McCulloch-Pitts Model of Neurons

This model is also known as **linear threshold gate** using **a linear step function** because it merely classifies the set of inputs into two different classes.
This model uses **hard-switch (step) activation function f** that makes the neuron active when **the weighted sum S** of **the input stimuli X** achieves **the threshold θ.**



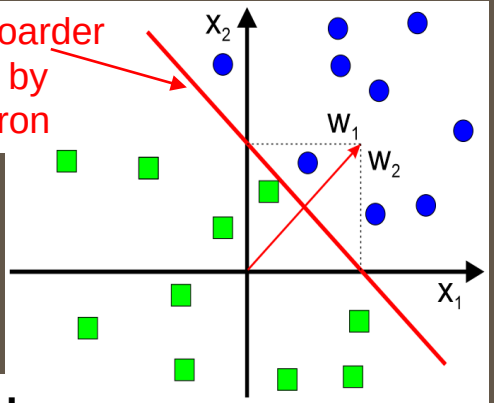
$$y = f(S) = \begin{cases} 0 & S < \theta \\ 1 & S \geq \theta \end{cases}$$

$$S = \sum_{k=1}^{K} x_k \cdot w_k$$

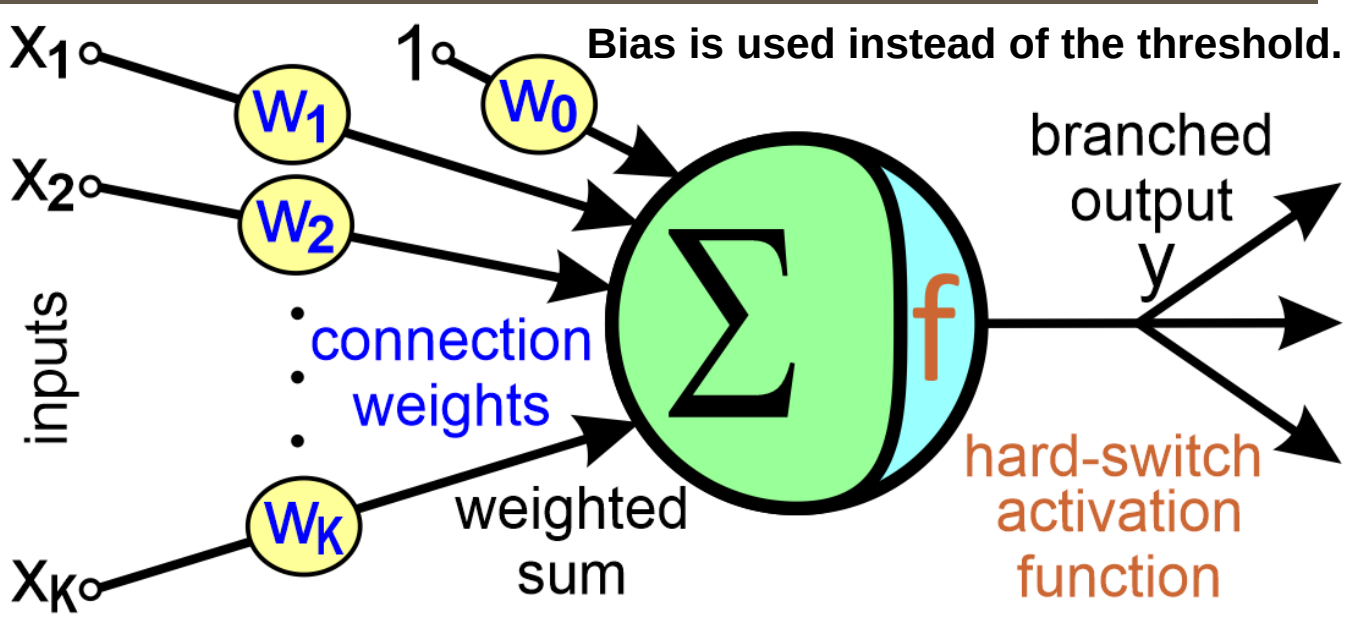
$$X = [x_1, \ldots, x_k]$$

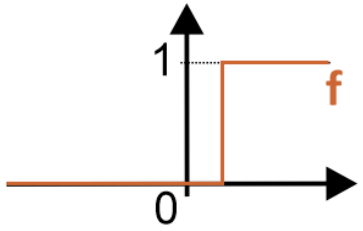
Hard-Switch Perceptron

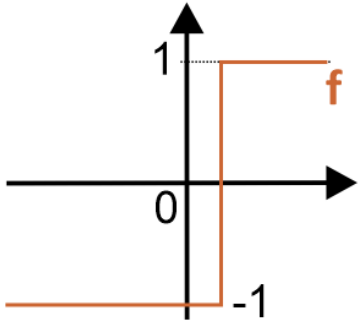
The decision boarder determined by the perceptron



This model originally employs **a step activation function** which serves as a **hard-switch** between two states: {0, 1} or {-1, 1} according to the used activation function f:







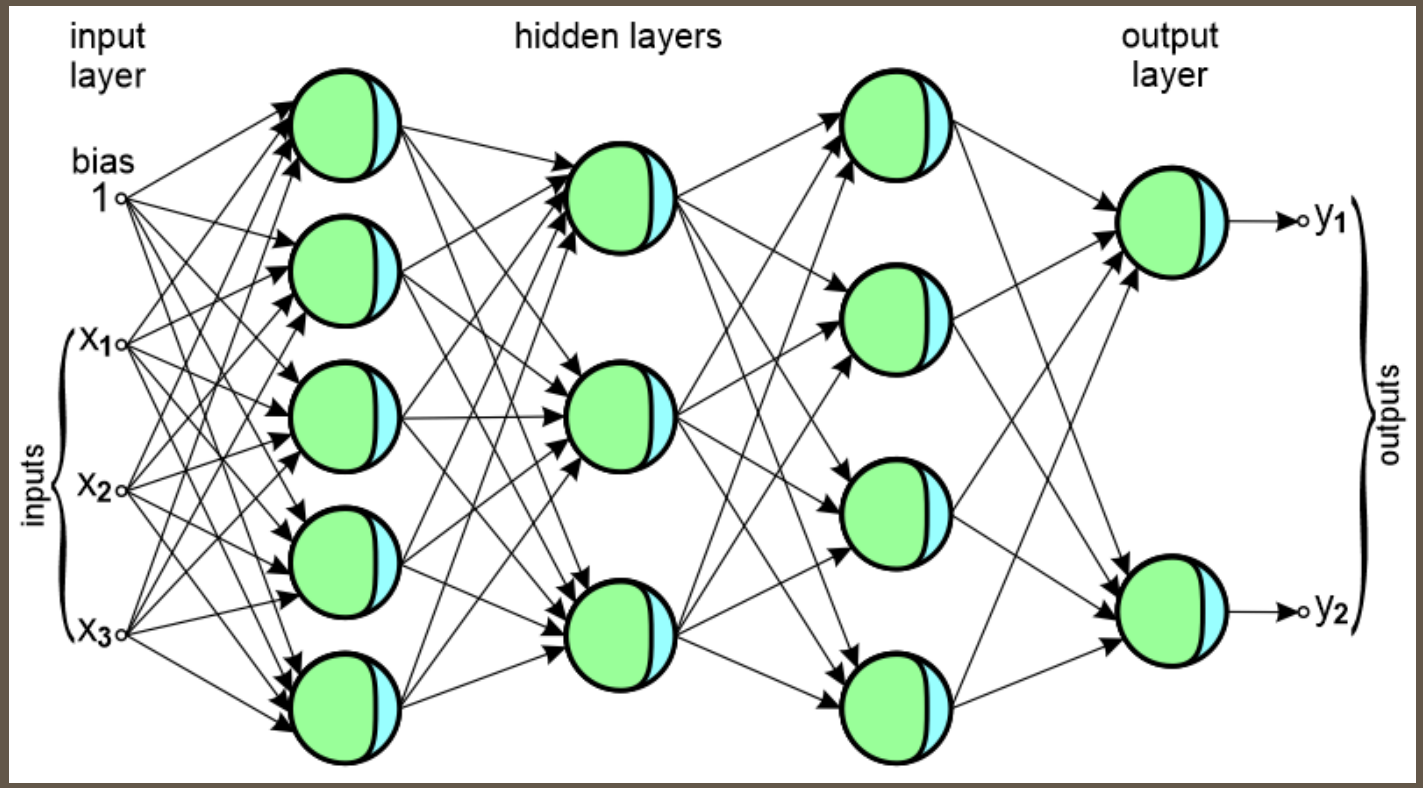
$$y = f(S) = \begin{cases} 0 & S < 0 \\ 1 & S \geq 0 \end{cases} \qquad S = \sum_{k=0}^{K} \mathbf{x}_k \cdot \mathbf{w}_k \qquad X = [\mathbf{x}_1, \ldots, \mathbf{x}_k] \qquad \mathbf{x}_0 = 1$$

Hard-Switch Perceptron Training

Supervised training of Hard-Switch Perceptron for a given training dataset consisting of training samples $\{(X_1, d_1), \ldots, (X_N, d_N)\}$, where d_n is the desired trained output value for the input training vector X_n, is defined as follows:

1. Randomly select small initial weights in the range of [-0.1, 0.1].
2. Stimulate the perceptron with the subsequent input training vector X_n, where $n = 1, \ldots, N$.
3. Compute a weighted sum S and an output value $y_n = f(S)$.
4. Compare the computed output value y_k with the desired trained output value d_n.
5. If $y_n \neq d_n$ then $\Delta w_k \mathrel{+}= (d_n - y_n) \cdot x_k$ else do nothing for the online training algorithm
   or compute $\Delta w_k = 1/N \cdot \sum_{n=1,\ldots,N} (d_n - y_n) \cdot x_k$ for the offline training algorithm.
6. Update the weights $w_k \mathrel{+}= \Delta w_k$ for all $k=0,\ldots,K$.
7. If the average iteration error $E = 1/N \cdot \sum_{n=1,\ldots,N} |d_n - y_n|$ is bigger than a user-specified error then start next iteration going to the step 2. The algorithm should also stop after processing some given maximum number of iterations.

Single and Multi-Layer Perceptrons

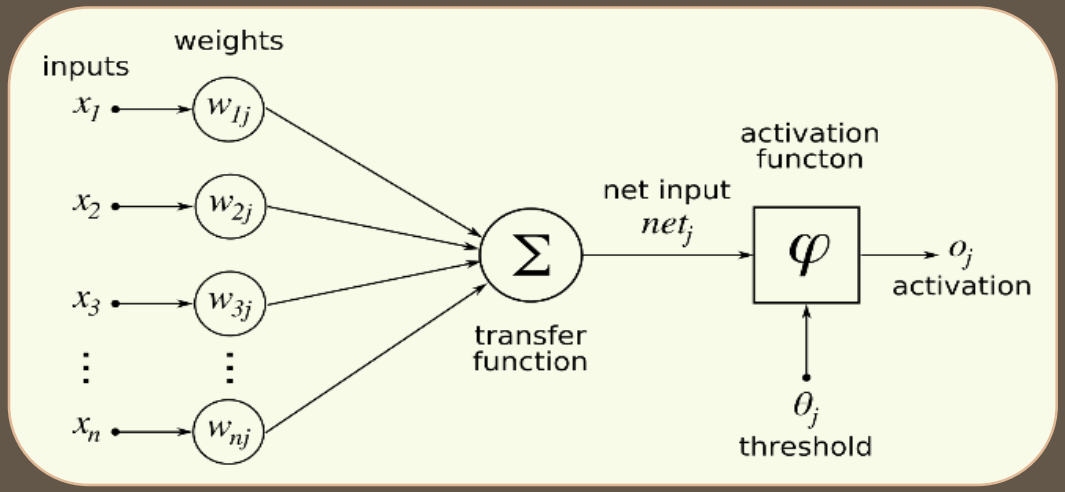


A group of perceptrons organized in a single layer can be used for the multi-classification which means the classification of input vectors into a few classes simultaneously.
Such a group of perceptrons is called a single-layer perceptron network which has a certain limitation of its adaptive capabilities.
For this reason, we usually use a multi-layer perceptron (MLP), i.e. the network that consists of several layers containing a various number of perceptrons.
The first layer is called input layer, the last one is called output layer, and all the layers between them are hidden as shown in the figure.

Brains and Neurons

Brains and real neurons inspired scientists to develop different models of artificial neurons and their networks.

The models of neurons using non-linear continuous activation functions (2nd generation) enable us building multilayer neural networks (e.g. MLP) and adapt such networks to more complex (non-linear) computational tasks.



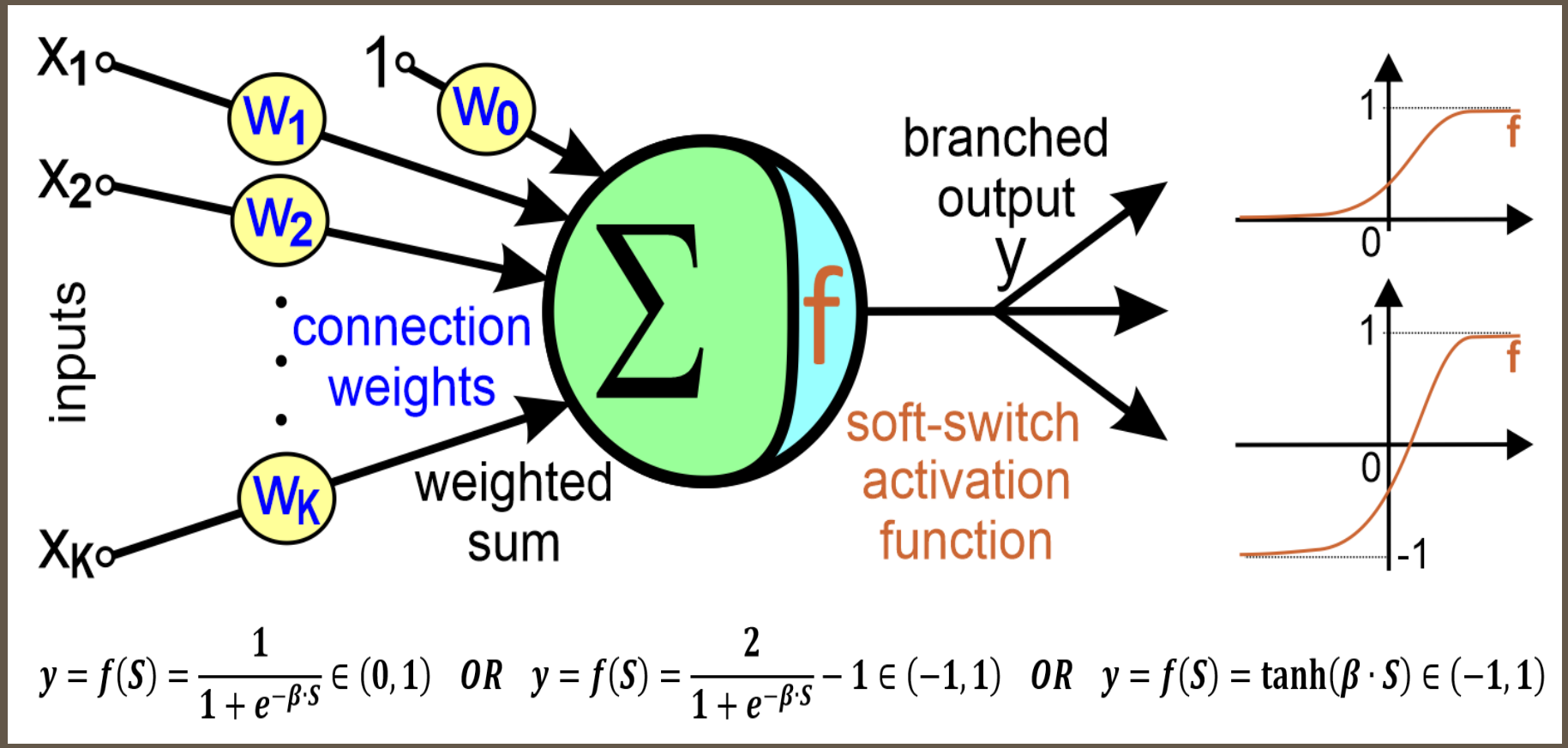
The use of hard-switch (step) activation functions limited the abilities of the first neurons, so mathematicians proposed to use non-linear soft-switch activation functions which were differentiable.

It allowed using gradient methods for adaptation (training) of such neurons.

This type of neuron models is the most frequently used one today, however, it has serious limitations that will be discussed later.

Soft-Switch Perceptron

This model employs a continuous sigmoid activation function which serves as a soft-switch between two states: (0, 1) or (-1, 1) according to the used function f:



$$y = f(S) = \frac{1}{1+e^{-\beta \cdot S}} \in (0,1) \quad OR \quad y = f(S) = \frac{2}{1+e^{-\beta \cdot S}} - 1 \in (-1,1) \quad OR \quad y = f(S) = \tanh(\beta \cdot S) \in (-1,1)$$

Delta Rule for Training

The delta rule uses the soft-switch neurons which activation functions are continuous to allow its differentiation. The delta is defined as the difference between the desired d_n and computed y_n outputs: $\delta_n = d_n - y_n$. This rule can be derivate as a result of the minimization of the mean square error function:

$$Q = \frac{1}{2}\sum_{n=1}^{N}(d_n - y_n)^2 \qquad where \qquad y_n = f(S) \qquad S = \sum_{k=0}^{K} x_k \cdot w_k$$

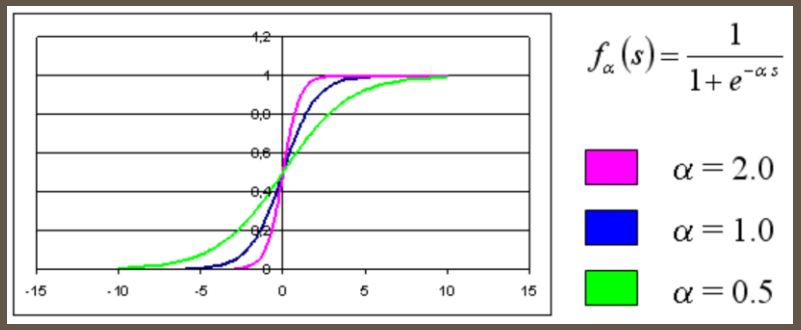
The correction of the weight for differentiable activation function f is computed after:

$$\Delta w_k = \eta \cdot \delta_n \cdot f'(S) \cdot x_k \qquad where \quad \delta_n = d_n - y_n$$

where f' is a derivative of the function f.

When the activation function $f(S) = \frac{1}{1+e^{-\alpha S}}$ is sigmoidal then we achieve the following expression for updating weight values:

$$\Delta w_k = \eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot x_k \qquad where \quad \delta_n = d_n - y_n$$



Introduction to Backpropagation Algorithm for Multilayer Perceptrons

The continuous, soft-switching nature of the sigmoid function allows it to be differentiable everywhere. This is necessary for several learning algorithms, such as Backpropagation or Convolutional Learning.

Because of the limited adaptive capabilities of a single-layer perceptron network, we use a multi-layer perceptron network (MLP) that consists of a few layers containing a various number of neurons.

Multi-layer perceptron cannot use linear soft-switch activation function because it can always be simplified to a single-layer linear perceptron network.

The MLP neural networks can be trained using Backpropagation Algorithm (BP), which overcomes the single-layer shortcoming pointed out by Minsky and Papert in 1969.

The BP algorithm is too slow to satisfy the machine learning needs, but it was rehabilitated later on (in 1989) when it became the learning engine of the far faster and the most popular Convolutional Deep Learning Neural Networks (CNN). Therefore, this algorithm is crucial for various neuronal architectures!

Backpropagation Algorithm

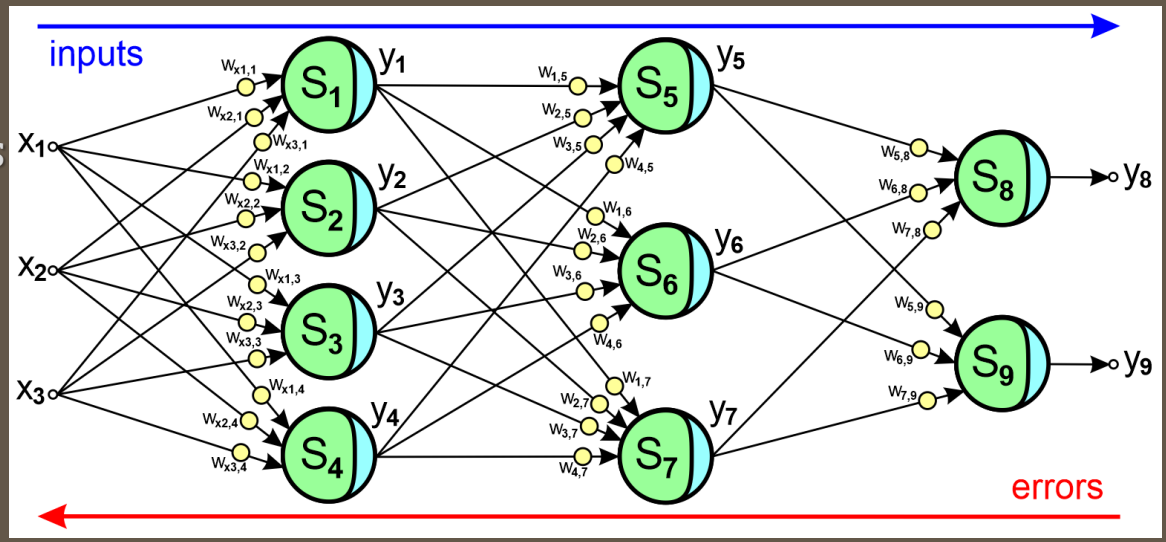
The **backpropagation algorithm (BP)** includes two main phases:

1. The input propagation phase propagates the inputs throughout all hidden layers to the output layer neurons. In this phase, neurons make summation of weighted inputs taken from the neurons in the previous layer.
2. The error propagation phase propagates back the errors (delta values) computed on the outputs of the neural network. In this phase, neurons make summation of weighted errors (delta values) taken from the neurons in the next layer.

The computed corrections of weights are used to update weights after:

- the computed corrections immediately after their computation during the online training,
- the average value of all computed corrections of each weight after finishing the whole training cycle for all training samples during the offline (batch) training.

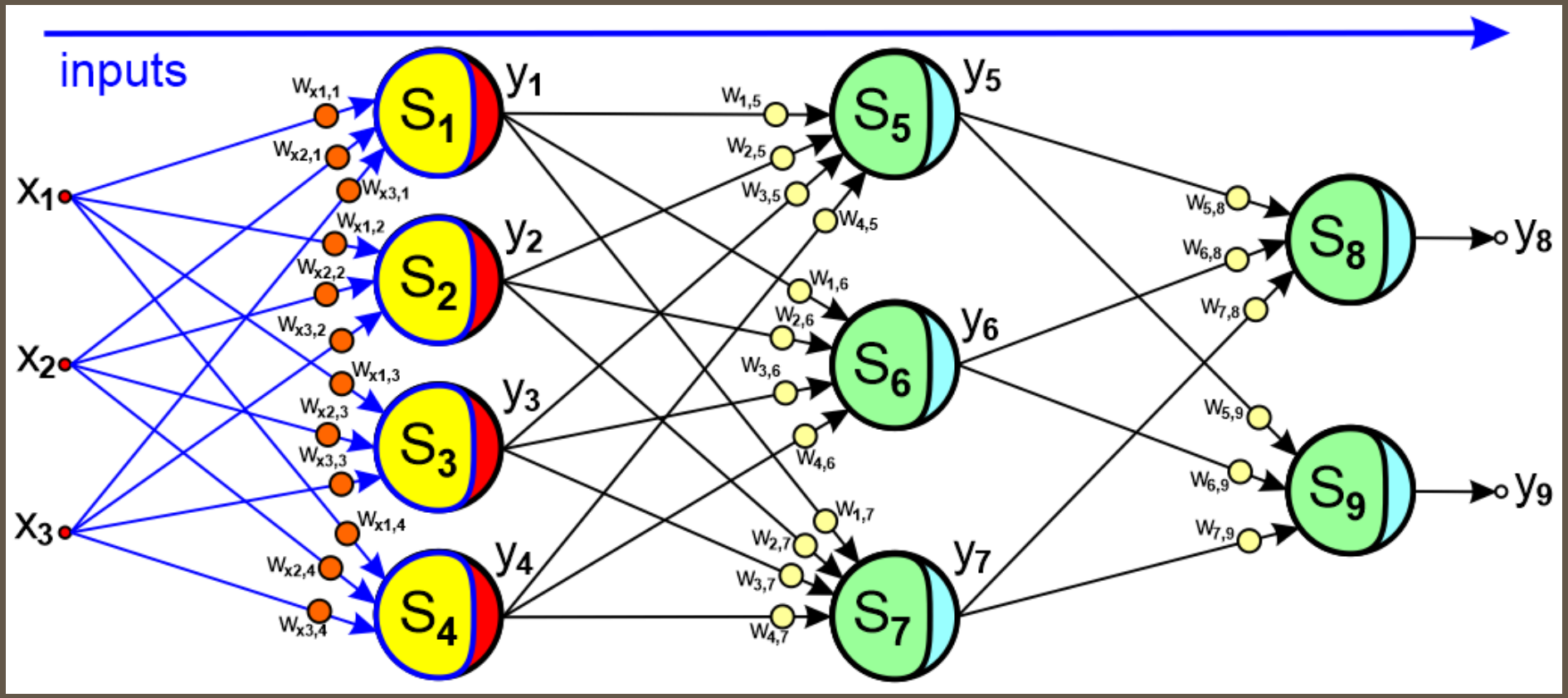
This algorithm is executed until the mean square error Q computed for all training samples is less than the desired value or to a given maximum number of cycles.



Backpropagation Algorithm

First, the inputs x_1, x_2, x_3 stimulate neurons in the first hidden layer.
The neurons compute weighted sums S_1, S_2, S_3, S_4, and output values y_1, y_2, y_3, y_4 that become inputs for the neurons of the next hidden layer:

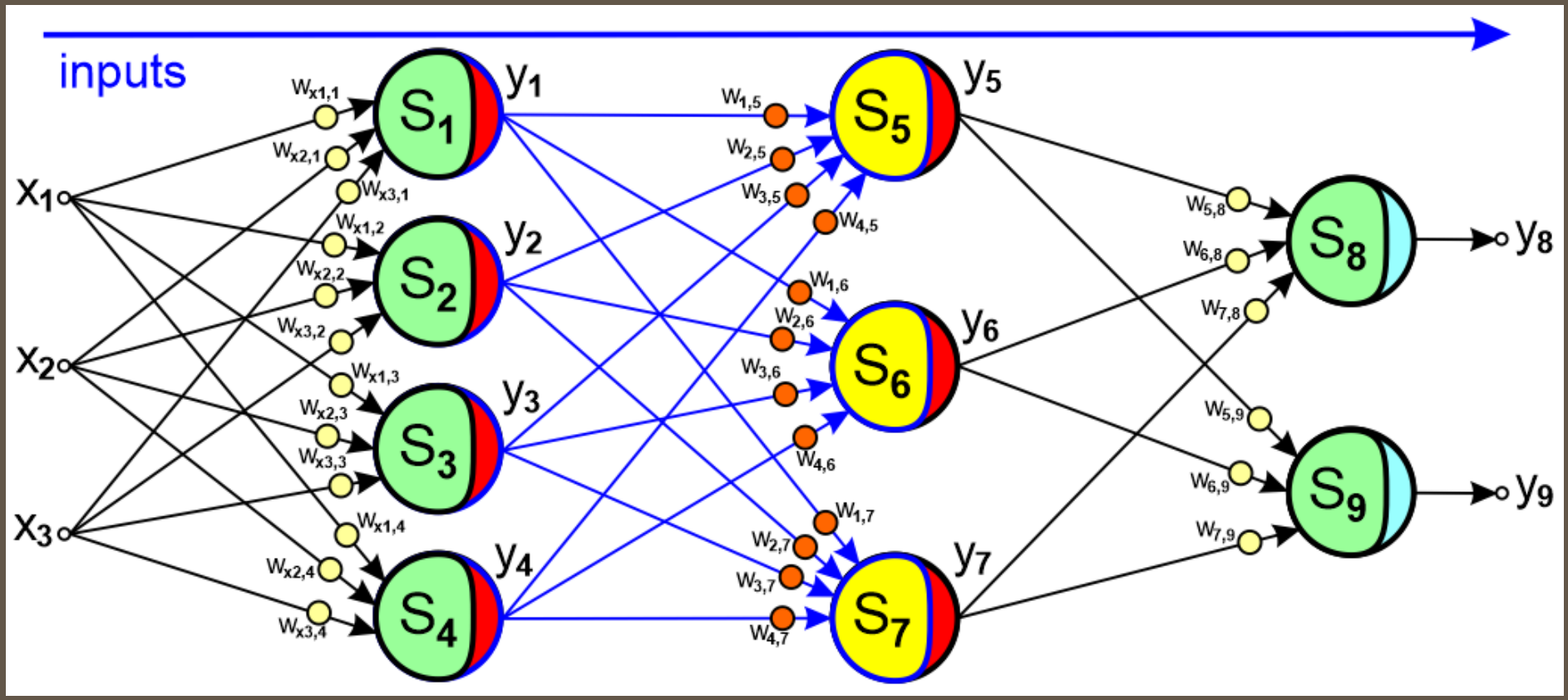
$$S_n = \sum_{k=1}^{3} x_k \cdot w_{x_k,n} \qquad y_n = f(S_n)$$



Backpropagation Algorithm

Second, the outputs y_1, y_2, y_3 ,y_4 stimulate neurons in the second hidden layer. The neurons compute weighted sums S_5, S_6, S_7, and output values y_5, y_6, y_7 that become inputs for the neurons of the output layer:

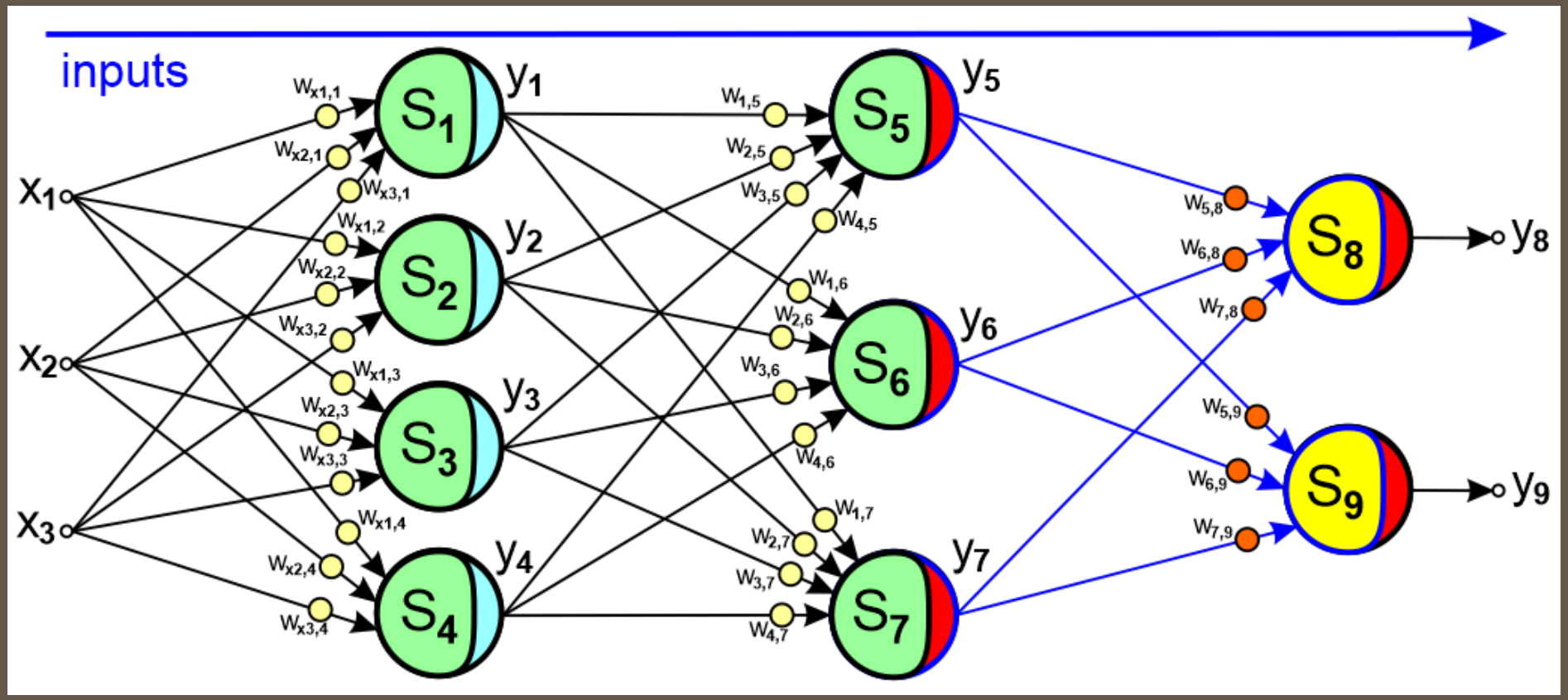
$$S_n = \sum_{k=1}^{4} y_k \cdot w_{k,n} \qquad y_n = f(S_n)$$



Backpropagation Algorithm

Finally, the outputs y_5, y_6, y_7 stimulate neurons in the output layer.
The neurons compute weighted sums S_8 and S_9, and output values y_8, y_9 that are the outputs of the neural network as well:

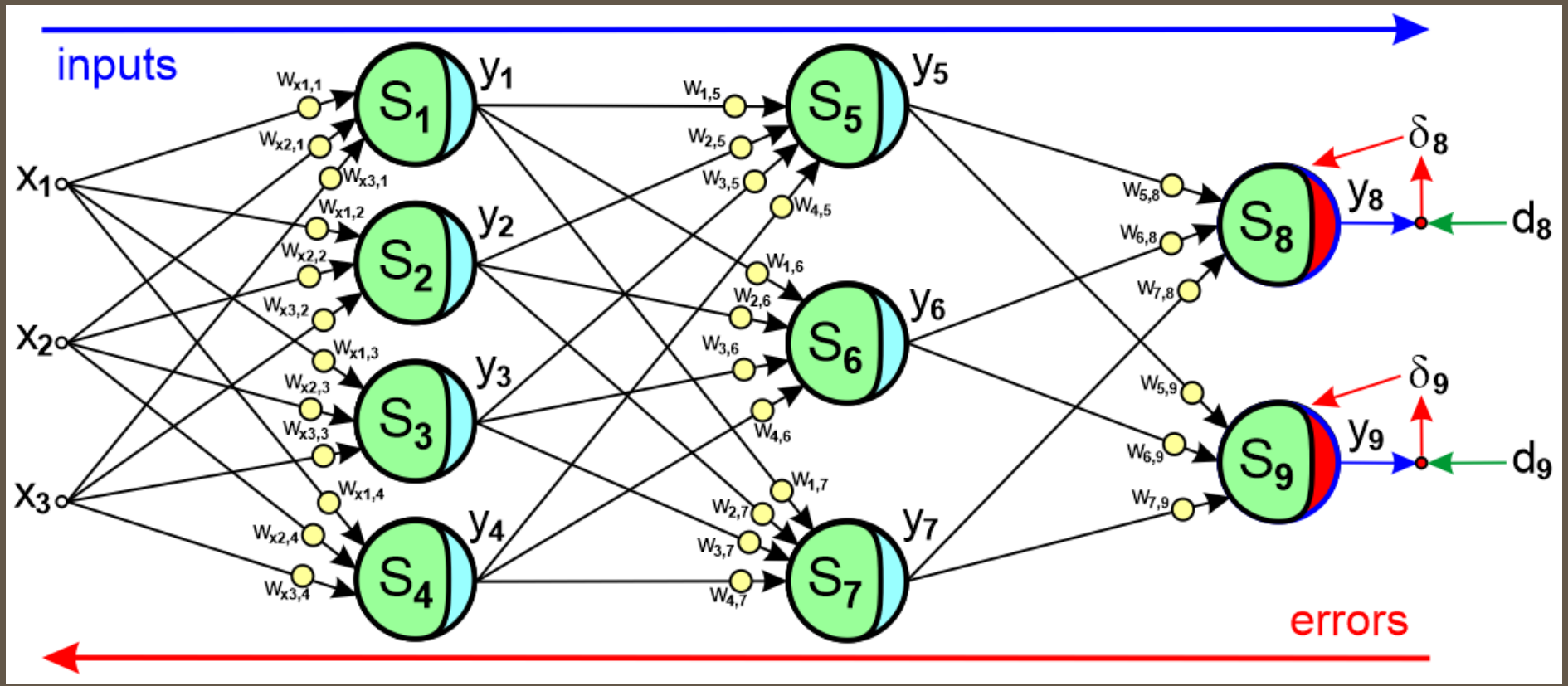
$$S_n = \sum_{k=5}^{7} y_k \cdot w_{k,n} \qquad y_n = f(S_n)$$



Backpropagation Algorithm

Next, the outputs y_8, y_9 are compared with the desired outputs d_8, d_9 and the errors δ_8, δ_9 are computed. These errors will be propagated back in order to compute corrections of weights from the connected inputs neurons.

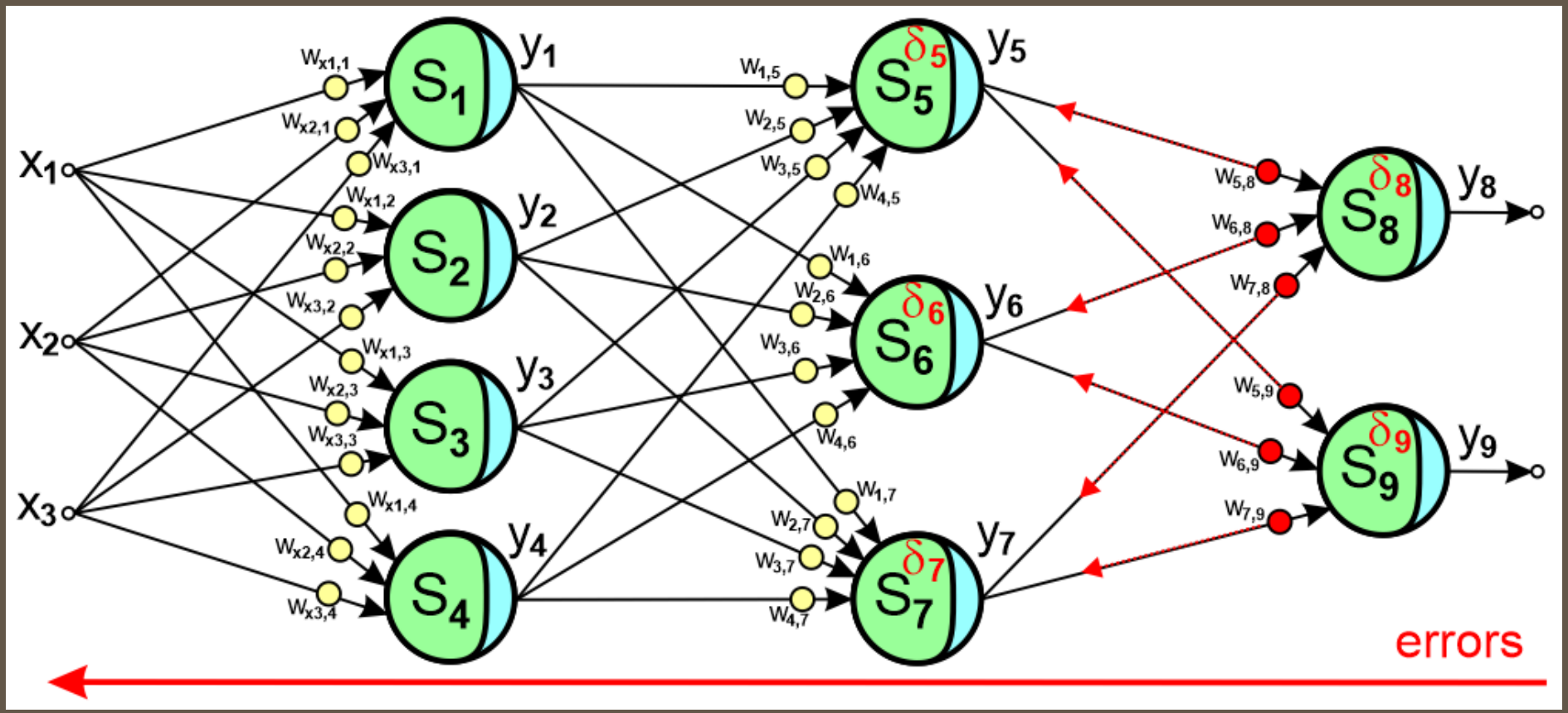
$$\delta_n = d_n - y_n$$



Backpropagation Algorithm

The errors δ_8 and δ_9 are used for corrections of the weights of the inputs connections y_5, y_6, y_7, and propagated back along the input connections to the neurons of the previous layer in order to compute their errors δ_5, δ_6, δ_7:

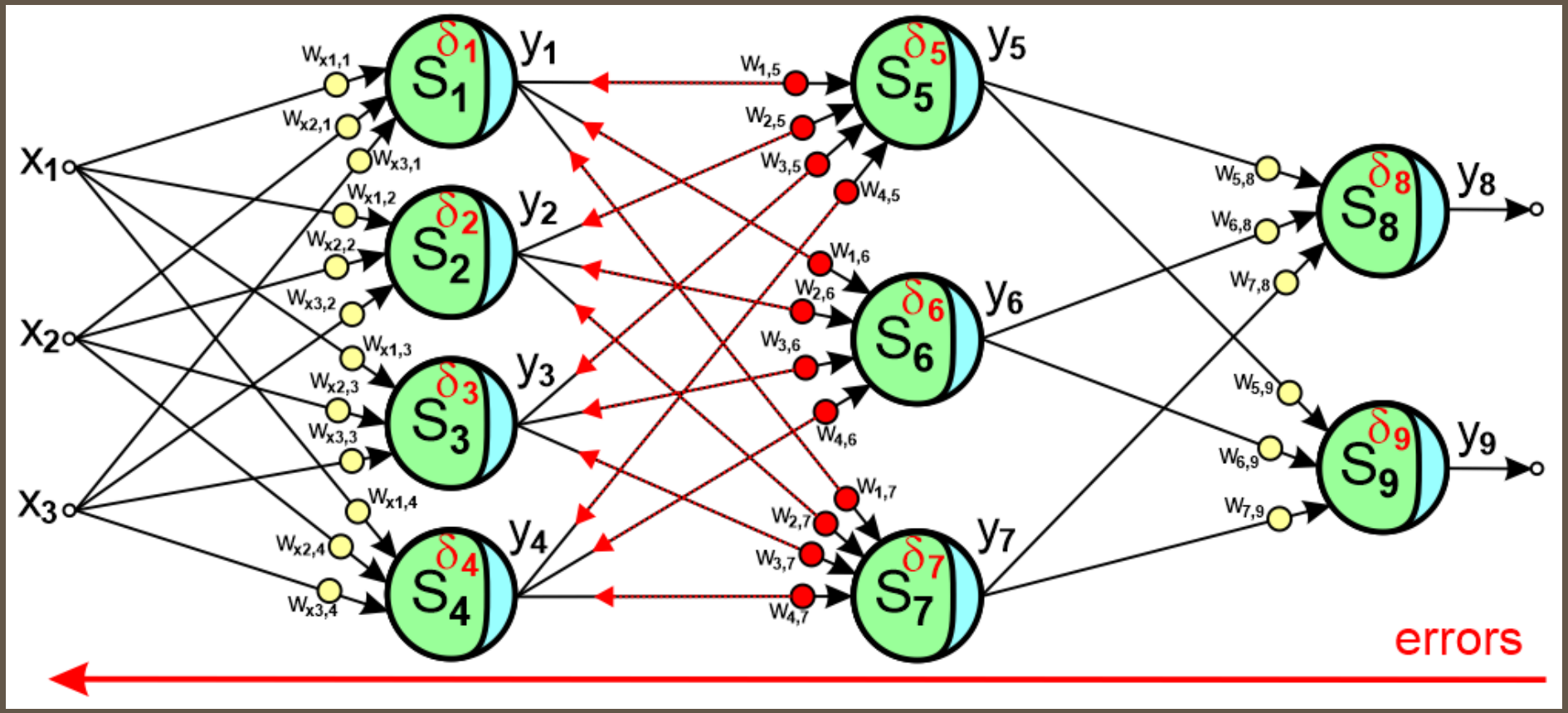
$$\Delta w_{k,n} = -\eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot y_k \qquad \delta_k = \sum_{n=8}^{9} \delta_n \cdot w_{k,n} \cdot (1 - y_n) \cdot y_n$$



Backpropagation Algorithm

Next, the errors δ_5, δ_6, and δ_7 are used for corrections of the weights of the inputs connections y_1, y_2, y_3, y_4, and propagated back along the input connections to the neurons of the previous layer in order to compute their errors δ_1, δ_2, δ_3, δ_4:

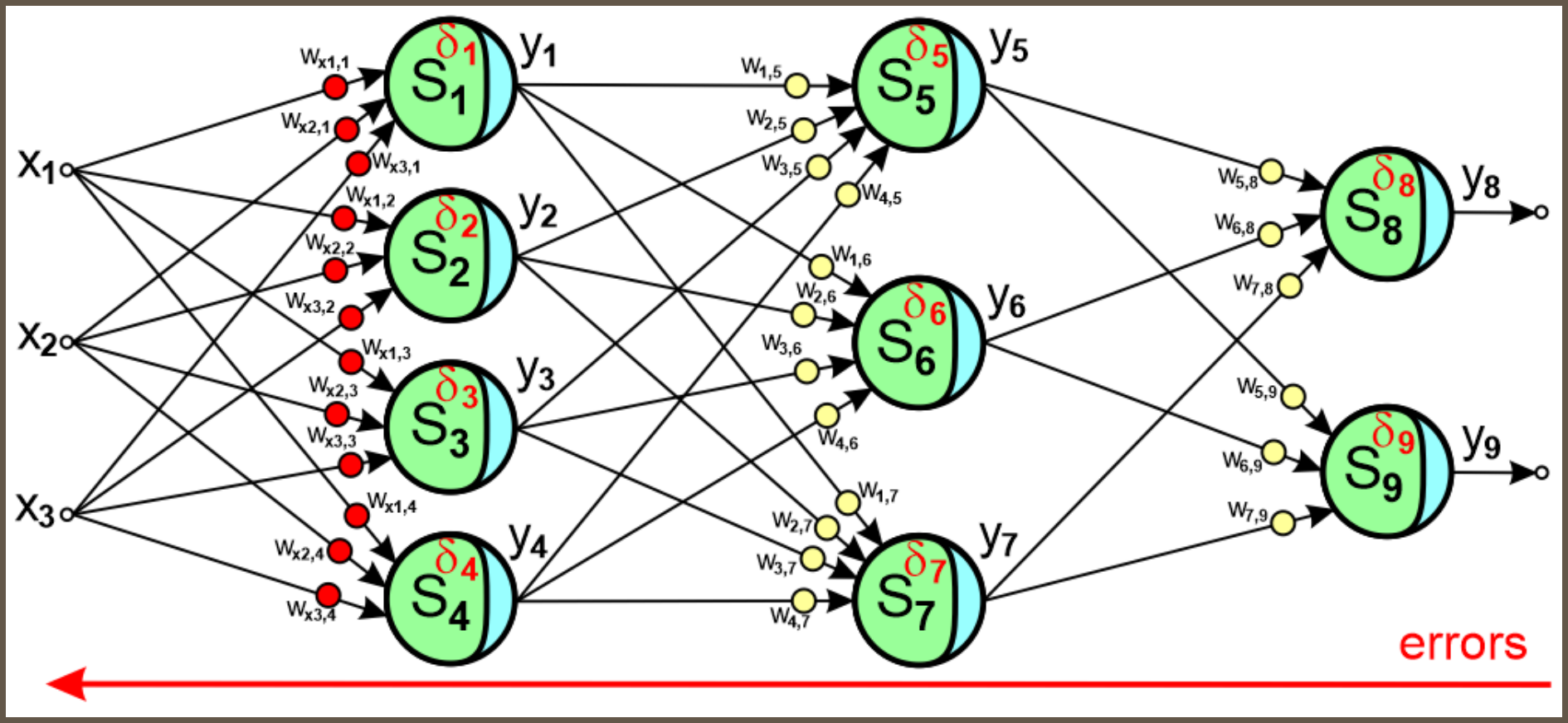
$$\Delta w_{k,n} = -\eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot y_k \qquad \delta_k = \sum_{n=5}^{7} \delta_n \cdot w_{k,n} \cdot (1 - y_n) \cdot y_n$$



Backpropagation Algorithm

Finally, the errors δ_1, δ_2, δ_3, δ_4 are used for corrections of the weights of the inputs x_1, x_2, x_3:

$$\Delta w_{k,n} = -\eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot y_k$$



Initialization and Training Parameters

The number of hidden layer neurons should be higher rather than lower to allow the network to create the representation of various features.
However, for simple problems, one or two hidden layers may be sufficient.

The numbers of neurons in the following layers usually decreases. They can also be fixed experimentally or using evolutional or genetic approaches that will be discussed later during these lectures and implemented during the laboratory classes.

Initialization of weights is accomplished by setting each weight to a low-valued random value selected from the pool of random numbers, say in the range from -1 to +1, or even smaller from -0.1 to +0.1.

The learning rate η should be adjusted stepwise ($\eta < 1$), considering stability requirements (we typically start from $\eta = 0.1$). However, since convergence is usually rather fast when the error becomes very small, it is advisable to reinstate η to its initial value before proceeding. We have many strategies and methods how to change this crucial parameter over the training process.

In order to avoid the BP algorithm from getting stuck (learning paralysis) at a local minimum or from oscillating the modification of learning rate should be employed.

Vanishing Gradient Problem

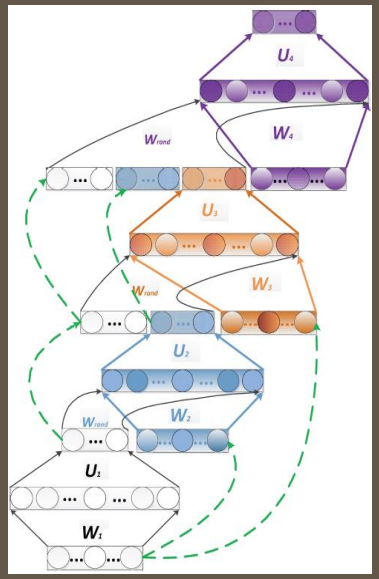
When using gradient-based learning strategies for many layers (e.g. MLPs) we usually come across the problem of vanishing gradients, because derivatives are always in range of [0, 1], so their multiple multiplications lead to very small numbers producing very small changes of weights in the neuron layers that are far away from the output of the MLP network.

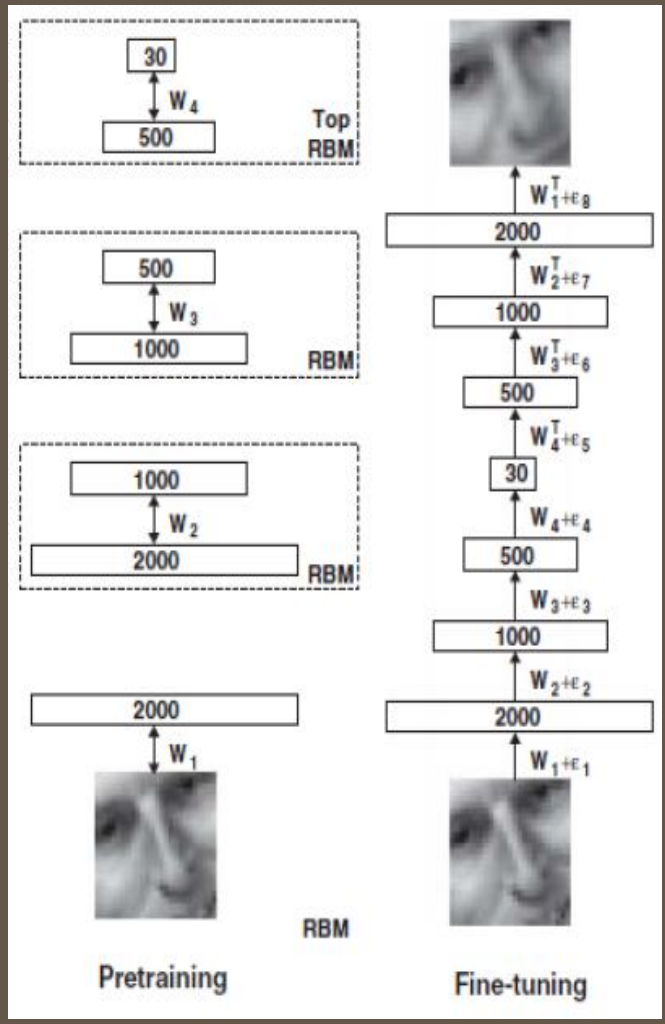
This problem can be solved using pre-training and fine-tuning strategy, which first trains the model layer after layer in the unsupervised way (e.g. using deep auto-encoder) and then we use backpropagation algorithm to fine-tune the network.

Hinton, Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. Science 2006

Hence, if we like to create a deep multilayer MLP topology, we have to deal with the problem of vanishing gradient problem.

To overcome this problem, we should construct the deep structure gradually. This will be one of the goals of our laboratory classes.





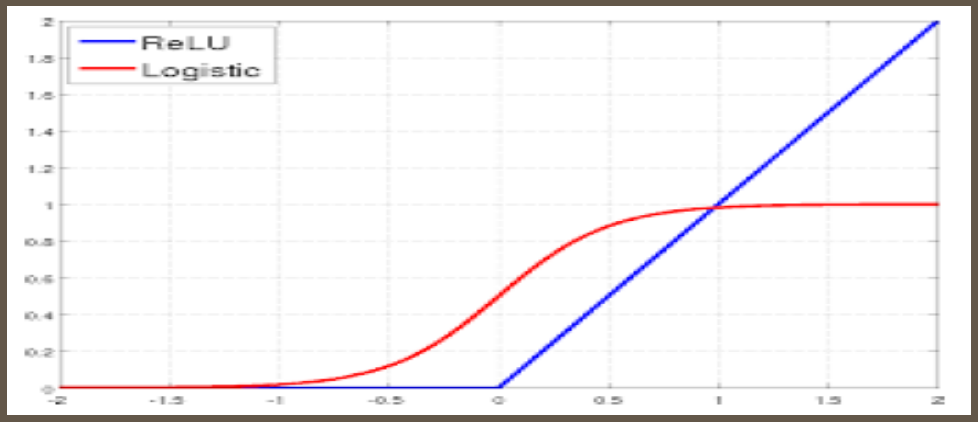
Rectified Linear Units (ReLU)

Rectified Linear Units (ReLU) eliminates the problem of vanishing gradients (i.e. derivatives are always in range of [0, 1], so their multiple multiplications lead to very small numbers producing very small changes of weights in the neuron layers that are far away from the output of the MLP network) when we use many hidden layers (e.g. in deep neural networks).

ReLU units are defined as: $f(S) = \max(0, S)$ instead of using the logistic function.

The strategy using ReLU units is based on training of robust features thanks to sparse (less frequent) activations of these units because when the function value is equal to 0 then we do not propagate the signal to the connected neurons and we do not need to propagate the delta back throughout such neurons as well during backpropagation.

The other outcome of using ReLU is that the training process is also typically faster.



Nair, Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. ICML 2010

Overcome Training Difficulties of BP

In order to overcome training difficulties of backpropagation algorithm we can use:

- Bias - an extra constant input (say $x_0=1$) that is weighted ($w_{0,n}$) and somehow resembles the threshold used in hard-switch neuron models.
- Momentum – that usually reduces the tendency to instability and avoids fast fluctuations ($\mathbf{0} < \boldsymbol{\alpha} < \mathbf{1}$), but it may not always work or could harm convergence:

$$\Delta \mathbf{w}_{k,n}^{p} = \boldsymbol{\alpha} \cdot \Delta \mathbf{w}_{k,n}^{p-1} + \boldsymbol{\eta} \cdot \boldsymbol{\delta}_n \cdot \boldsymbol{f}' \left(\sum_{k=0}^{K} \mathbf{x}_k \cdot \mathbf{w}_k \right) \cdot \mathbf{x}_k = \boldsymbol{\alpha} \cdot \Delta \mathbf{w}_{k,n}^{p-1} + \boldsymbol{\eta} \cdot \boldsymbol{\delta}_n \cdot (\mathbf{1} - \boldsymbol{y}_n) \cdot \boldsymbol{y}_n \cdot \mathbf{x}_k$$

- Smoothing – that is also not always advisable for the same reason:

$$\Delta \mathbf{w}_{k,n}^{p} = \boldsymbol{\alpha} \cdot \Delta \mathbf{w}_{k,n}^{p-1} + (\mathbf{1} - \boldsymbol{\alpha}) \cdot \boldsymbol{\delta}_n \cdot \boldsymbol{f}' \left(\sum_{k=0}^{K} \mathbf{x}_k \cdot \mathbf{w}_k \right) \cdot \mathbf{x}_k$$

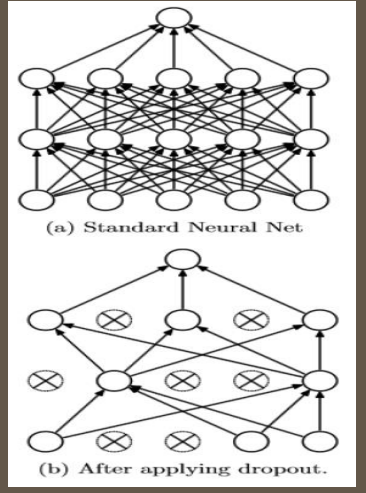
$$= \boldsymbol{\alpha} \cdot \Delta \mathbf{w}_{k,n}^{p-1} + (\mathbf{1} - \boldsymbol{\alpha}) \cdot \boldsymbol{\delta}_n \cdot (\mathbf{1} - \boldsymbol{y}_n) \cdot \boldsymbol{y}_n \cdot \mathbf{x}_k$$

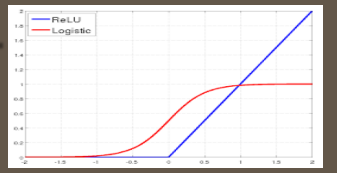
where p is the training period (cycle) of training samples.

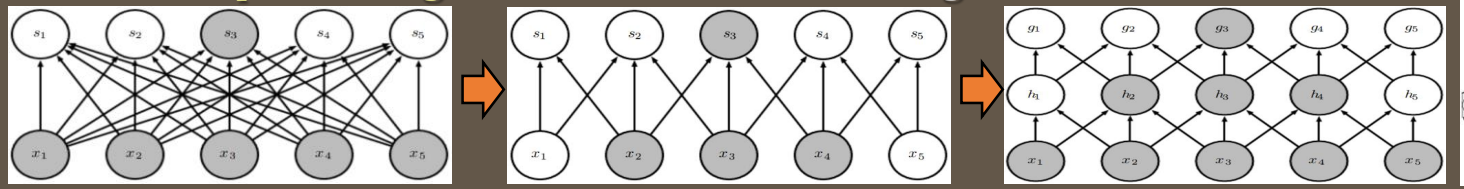
Overcome Convergence Problems of BP

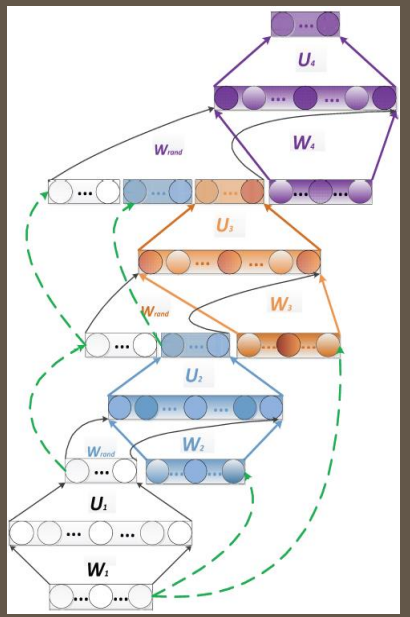
To overcome convergence problems of the backpropagation algorithm we can:

- Modifying the step size (learning rate η) during the adaptation process.
- Start the training process many times with various initial weights.
- Use various network architectures, e.g. change the number of layers or the number of neurons in these layers.
- Use a genetic algorithm or an evolutional approach to find a more appropriate architecture of a neural network.
- Switching between off-line and on-line training strategies because the off-line strategy is faster and more stable, but the on-line strategy better escapes local minima.
- Reduce the number of inputs to overcome the curse of dimensionality problem.
- Use sparse connections, not all-to-all between subsequent layers.
- Change the range of the sigmoid function from [0, 1] to [-1, 1].
- Freeze weights in a previously trained layer or subnetwork.
- Use cross-validation to avoid the problem of over-fitting.
- Use rectified linear units (ReLU), $f(S) = \max(0, S)$.
- Use dropout regularization technique.
- Use deep learning architectures and strategies.









K-fold Cross-Validation

Cross-Validation strategy allows us to use all available patterns for training and validation alternately during the training process.

„K-fold" means that we divide all training patterns into K disjoint more or less equinumerous subsets. Next, we train a selected model on K-1 subsets K-times and also test this model on an aside subset K-times.

The validation subset changes in the course of the next training steps:

5-FOLD	SUBSETS OF TRAINING PATTERNS				
STEPS	1	2	3	4	5
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST

K-fold Cross-Validation

We use different K parameters according to the number of training patterns:

K is usually small ($3 \leq K \leq 10$) for numerous training patters.

It lets us validate the model better if it is tested on a bigger number of training patterns.

It also reduces the number of training steps that must be performed.

K is usually big ($10 \leq K \leq N$) for less numerous training datasets, where N is the total number of all training patterns.

It allows us to use more patterns for training and achieve a better-fitted model.

5-FOLD	SUBSETS OF TRAINING PATTERNS				
STEPS	1	2	3	4	5
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST

10-FOLD	SUBSETS OF TRAINING PATTERNS									
STEPS	1	2	3	4	5	6	7	8	9	10
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
6	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN
7	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN
8	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN
9	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN
10	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST

N-fold Cross-Validation

N-folds Cross-Validation (one-leave-out strategy) is rarely used because the N-element dataset has to be trained N times. The following disadvantage is that we use only a single pattern in each step for validation of the whole model. Such a result is not representative of the entire collection and the CI model. This solution is sometimes used for tiny datasets.

N-FOLD	One-element subsets of the training patter set consisting of 20 patterns																			
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
6	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
7	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
8	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
9	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
10	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
11	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
12	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
13	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
14	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
15	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
16	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN
17	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN
18	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN
19	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN
20	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST

K-fold Cross-Validation Selection Strategies

The way of selection of the test patterns in each training step should be proportional and representative from each class point of view regardless of the cardinality of classes!
We have to consider how the training data are organized in the training dataset:

- Randomly
- Grouped by categories (classes)
- Ordered by values of their attributes
- Grouped by classes and ordered by values of their attributes
- In an unknown way

5-FOLD	SUBSETS OF TRAINING PATTERNS THAT ARE RANDOMLY ORDERED IN THE DATA SET																																					
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TEST

5-FOLD	SUBSETS OF TRAINING PATTERNS																																					
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST

5-FOLD	SUBSETS OF TRAINING PATTERNS														
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST

K-fold Cross-Validation Random Strategy

The test patterns can also be selected randomly with or without repetition:

5-FOLD	SUBSETS OF TRAINING PATTERNS																																						
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN
2	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST
3	TEST	TRAIN	TRAIN	TRAIN	TEST	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN
4	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN

5-FOLD	SUBSETS OF TRAINING PATTERNS THAT ARE RANDOMLY ORDERED IN THE DATA SET																																						
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN
2	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST
3	TEST	TRAIN	TRAIN	TRAIN	TEST	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN
4	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN

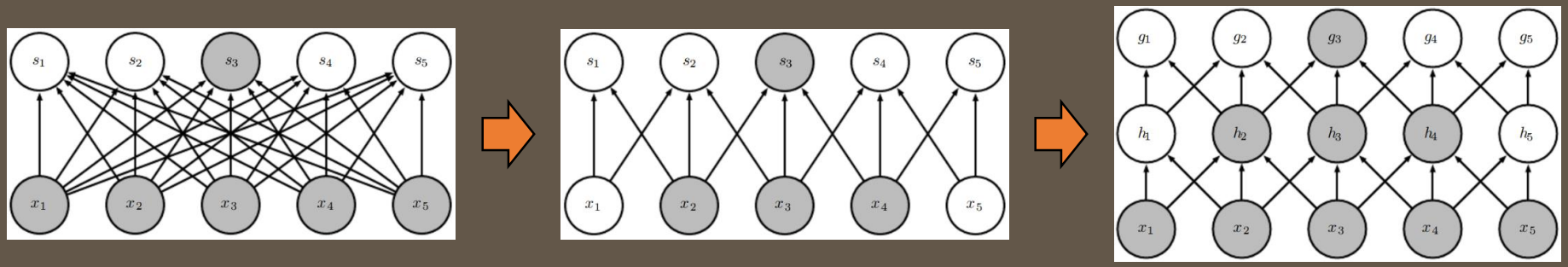
The choice between various options should be made on the basis of the initial order or disorder of patterns of all classes in the dataset to achieve representative selection of the test patterns used for the validated model.

Patterns used for validation should not be repeated in successive test groups, only that we use a less reliable and simpler approach to random choosing of validation patterns.

Deep Learning Strategies and Networks

Deep learning strategies assume the ability to:

- update only a selected part of neurons (drop-out) that respond best to the given input data, so the other neurons and their parameters (e.g. weights, thresholds) are not updated,
- avoid connecting all neurons between successive layers, so we do not use all-to-all connection strategy known and commonly used in MLP and other networks, but we try to allow neurons to specialize in recognizing subpatterns that can be extracted from the limited subsets of inputs,

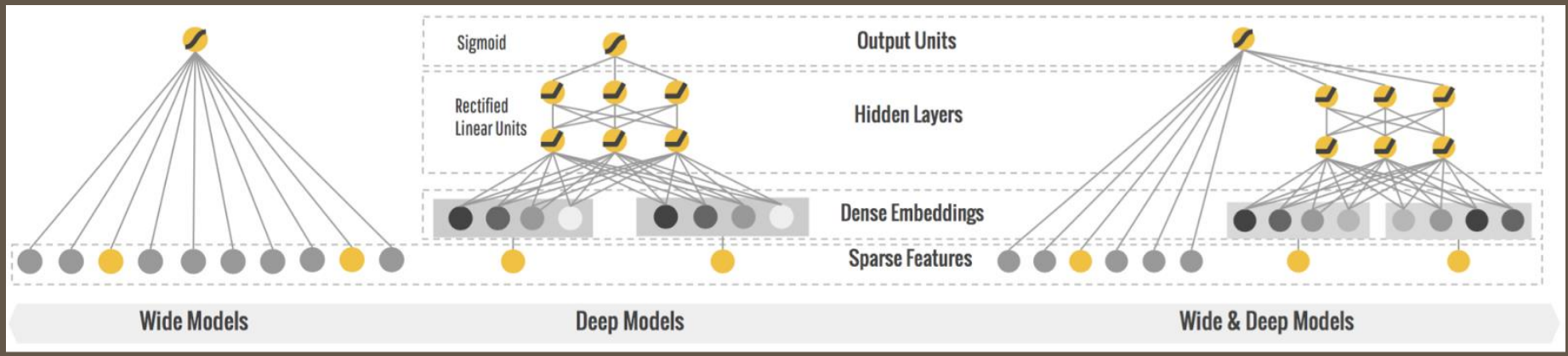


- create connections between various layers and subnetworks, not only between successive layers
- use many subnetworks that can be connected in different ways in order to allow neurons from these subnetworks to specialize in defining or recognizing of limited subsets of features or subpatterns,
- let neurons specialize and not overlap represented regions and represent the same features or subpatterns.

Wide (Broad) versus Deep Neural Networks

Wide (Broad) Neural Networks assume to:

- Add neurons in one hidden layer until the results of training will be satisfactory.
- During this kind of training we sometimes freeze the weights of the already added neurons and adapt only the currently added neuron. This makes the training process faster, however not necessarily better.
- Wide (broad) networks adapt usually much faster than deep networks.

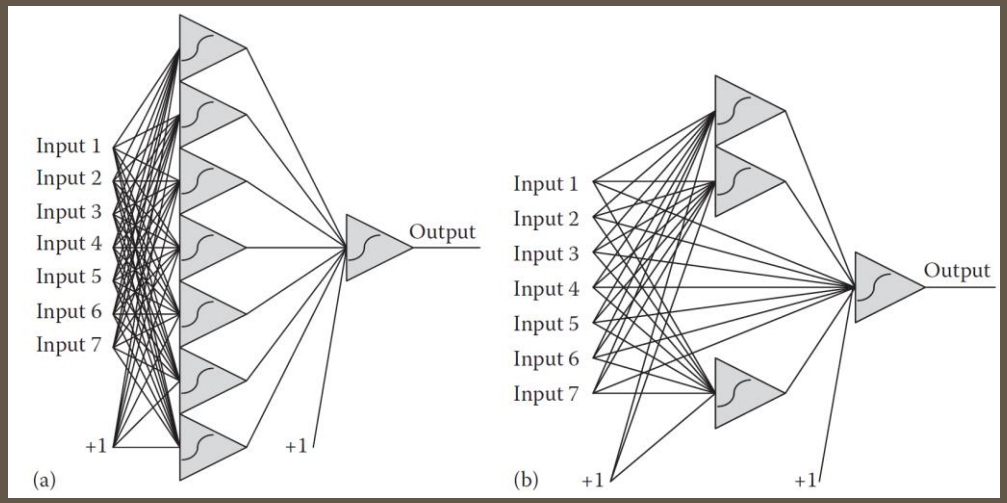


- We can develop:
  - Wide neural network models (adding new neurons in the same layer),
  - Deep neural network models (adding new layers of neurons in the various layer),
  - Wide & deep network models (combining these two approaches),
  - Subnetworks that specialize in representation of a limited subset of features and next combined these subnetworks into one big neural network.

BMLP and FCC Neural Networks

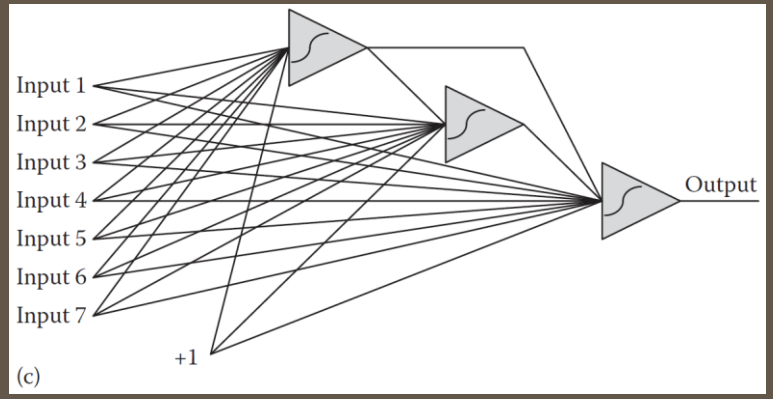
BMLP Neural Networks assume to:

- connect neurons not only to the previous layer neurons but additionally to all inputs:



Fully Connected Cascade (FCC) Neural Networks assume to:

- add new neurons in the next hidden layers (each hidden layer consists only from one neuron) and connect them to all inputs and all neurons from the previous layers.

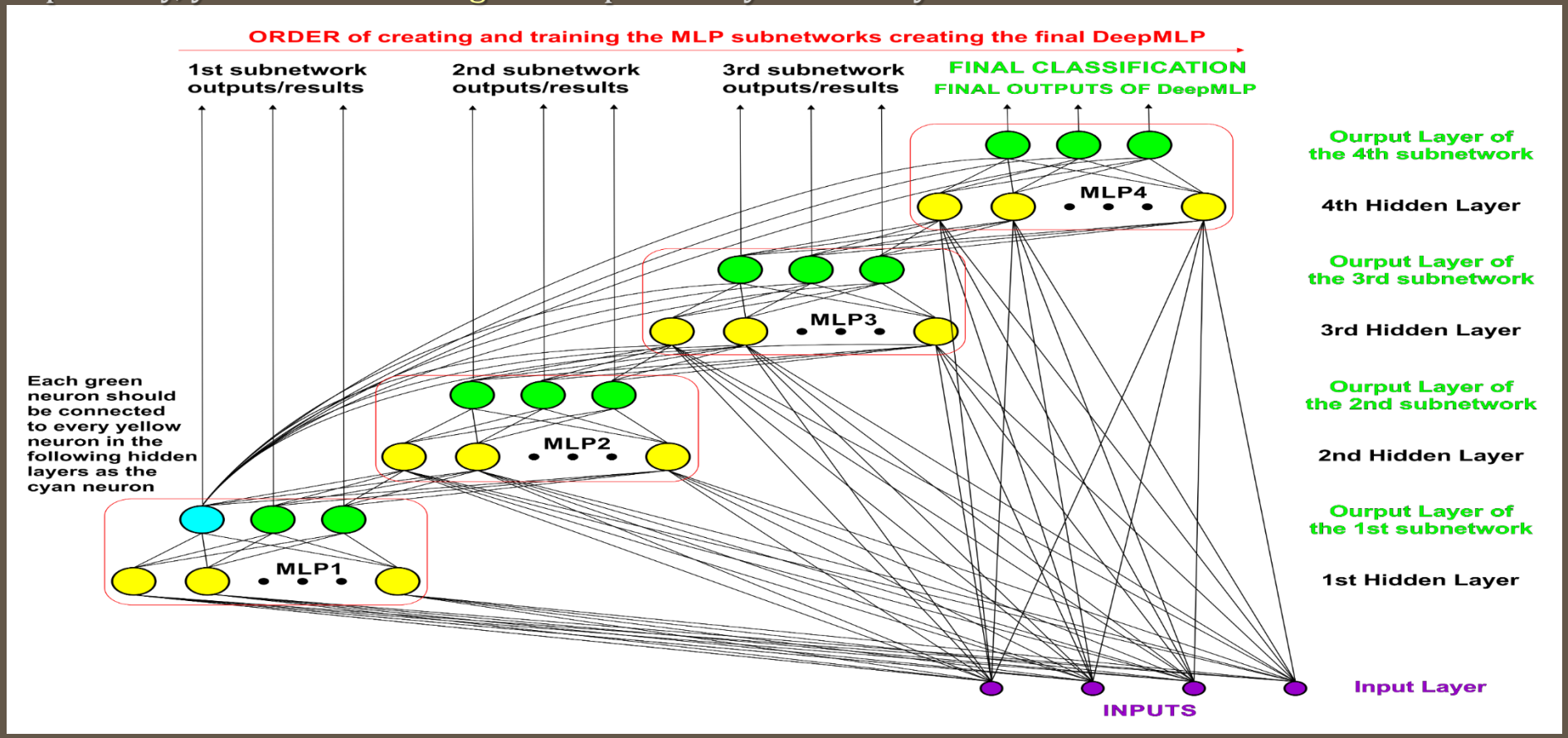


Combined Deep Structures of Neural Networks

Gradually developed Deep Neural Networks assume to:

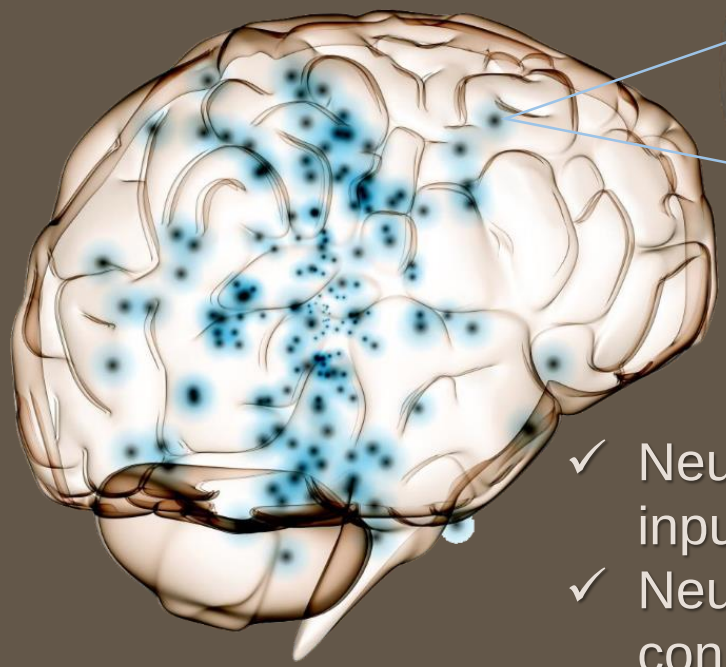
1. Train small neural network architecture (usually with one hidden layer) until this network decreases its training error.
2. Create the next subnetwork and connect it again to all raw inputs and additionally to outputs of the previously created and trained subnetworks which weights can be frozen or left for next adaptation together with the newly created subnetwork.
3. Repeat step 2 until the network achieves the satisfactory low level of the error function.

Optionally, you can freeze weights in a previously trained layer or subnetwork.



Plastic Associative Strategies of Real Neurons

We can find a solution in the brain structures where data are stored together with their relations.

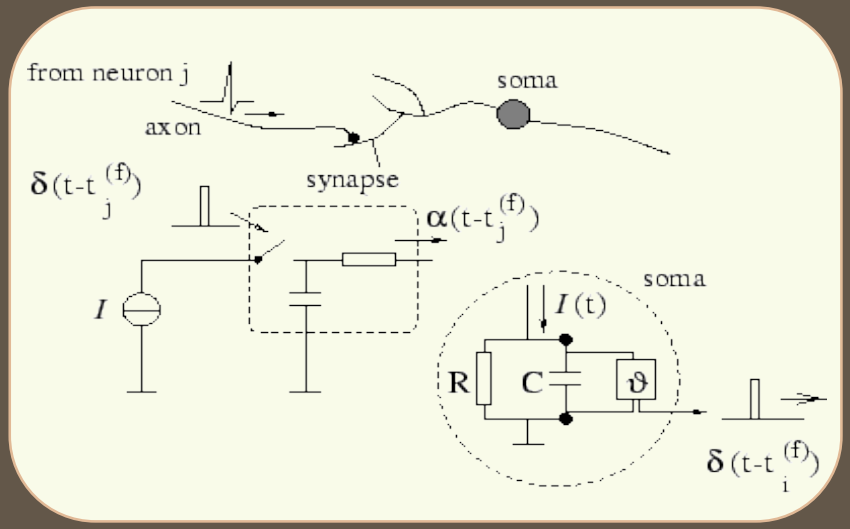


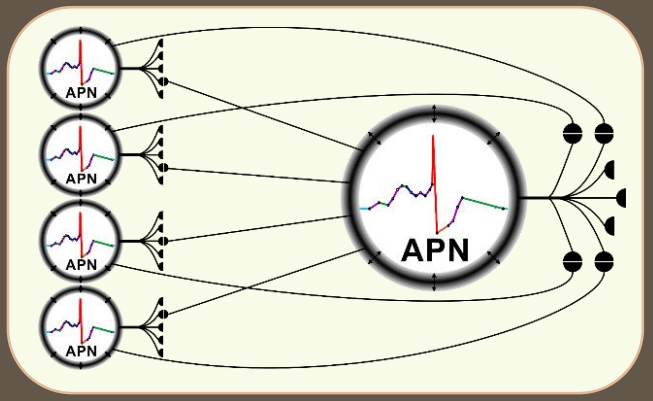
- ✓ Neurons can represent any subset of input data combinations which activate them.
- ✓ Neuronal plasticity processes automatically connect neurons and reinforce connections which represent related data and objects.

Let us use the biologically optimized solution!

Neuron Models of the 3rd and 4th Generations

3. **The spiking models of neurons** enriched this model with the implementation of the approach of time which is very important during stimuli integration and subsequent processes modeling.





4. **The associative pulsing models (APN) of neurons** produce a series of pulses (spikes) in time which frequency determines the association level. Moreover, they enrich the model with the automatic plastic mechanism which let neurons to conditionally connect and configure associative neural structures representing data, objects, their sequences, and relationships between them.

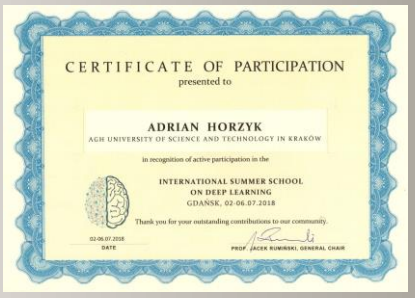
Bibliography and Literature

1. Nikola K. Kasabov, ***Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence***, In Springer Series on Bio- and Neurosystems, Vol 7., Springer, 2019.
2. Ian Goodfellow, Yoshua Bengio, Aaron Courville, ***Deep Learning***, MIT Press, 2016, ISBN 978-1-59327-741-3 or PWN 2018.
3. Holk Cruse, ***Neural Networks as Cybernetic Systems***, 2nd and revised edition
4. R. Rojas, ***Neural Networks***, Springer-Verlag, Berlin, 1996.
5. ***Convolutional Neural Network*** (Stanford)
6. ***Visualizing and Understanding Convolutional Networks***, Zeiler, Fergus, ECCV 2014
7. **IBM: https://www.ibm.com/developerworks/library/ba-data-becomes-knowledge-1/index.html**
8. **NVIDIA: https://developer.nvidia.com/discover/convolutional-neural-network**
9. **A. Horzyk**, J. A. Starzyk, J. Graham, ***Integration of Semantic and Episodic Memories***, IEEE Transactions on Neural Networks and Learning Systems, Vol. 28, Issue 12, Dec. 2017, pp. 3084 - 3095, 2017, DOI: 10.1109/TNNLS.2017.2728203.
10. **A. Horzyk**, J.A. Starzyk, ***Multi-Class and Multi-Label Classification Using Associative Pulsing Neural Networks***, IEEE Xplore, In: 2018 IEEE World Congress on Computational Intelligence (WCCI IJCNN 2018), 2018, (in print).
11. **A. Horzyk**, J.A. Starzyk, ***Fast Neural Network Adaptation with Associative Pulsing Neurons***, IEEE Xplore, In: 2017 IEEE Symposium Series on Computational Intelligence, pp. 339 -346, 2017, DOI: 10.1109/SSCI.2017.8285369.
12. **A. Horzyk**, K. Gołdon, ***Associative Graph Data Structures Used for Acceleration of K Nearest Neighbor Classifiers***, LNCS, In: 27th International Conference on Artificial Neural Networks (ICANN 2018), 2018, (in print).
13. **A. Horzyk**, ***Deep Associative Semantic Neural Graphs for Knowledge Representation and Fast Data Exploration***, Proc. of KEOD 2017, SCITEPRESS Digital Library, pp. 67 - 79, 2017, DOI: 10.13140/RG.2.2.30881.92005.
14. **A. Horzyk**, ***Neurons Can Sort Data Efficiently***, Proc. of ICAISC 2017, Springer-Verlag, LNAI, 2017, pp. 64 - 74, ICAISC BEST PAPER AWARD 2017 sponsored by Springer.
15. **Horzyk, A.**, ***How Does Generalization and Creativity Come into Being in Neural Associative Systems and How Does It Form Human-Like Knowledge?***, Elsevier, Neurocomputing, Vol. 144, 2014, pp. 238 - 257, DOI: 10.1016/j.neucom.2014.04.046.

Adrian Horzyk

horzyk@agh.edu.pl

Google: Horzyk



University of Science and Technology in Krakow, Poland