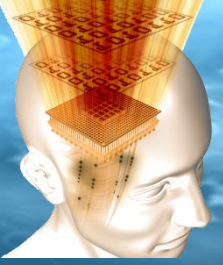
COMPUTATIONAL INTELLIGENCE

LABORATORY CLASSES

Implementation of MLP, Backpropagation with Cross-Validation,
using Wide and Deep Learning Strategies and Approaches

AGH

Adrian Horzyk



Implementation of Multi-Layer Perceptron (MLP) Neural Network

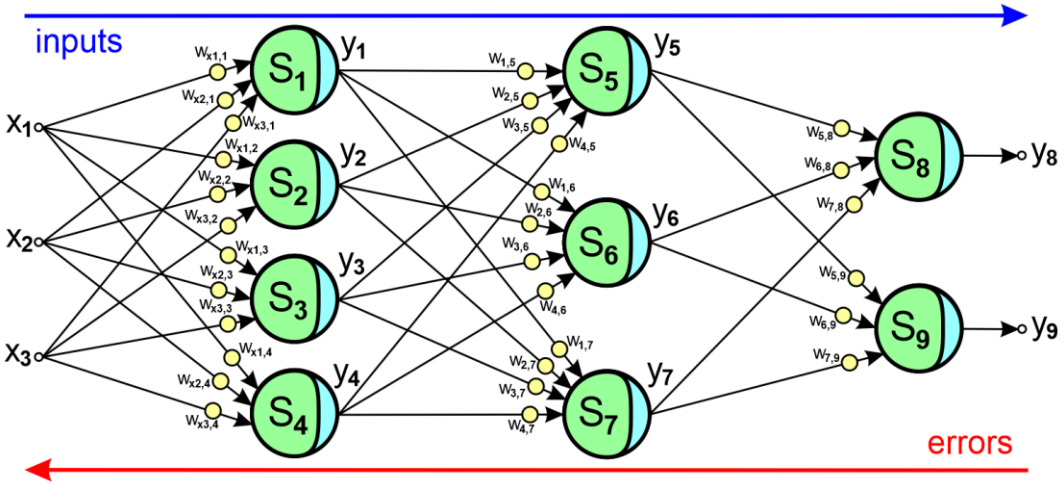
The MLP neural networks and the backpropagation training algorithm are the most popular and widely used in Computational Intelligence to be employed in many simple classification and regression models as well as in deep learning strategies and networks.

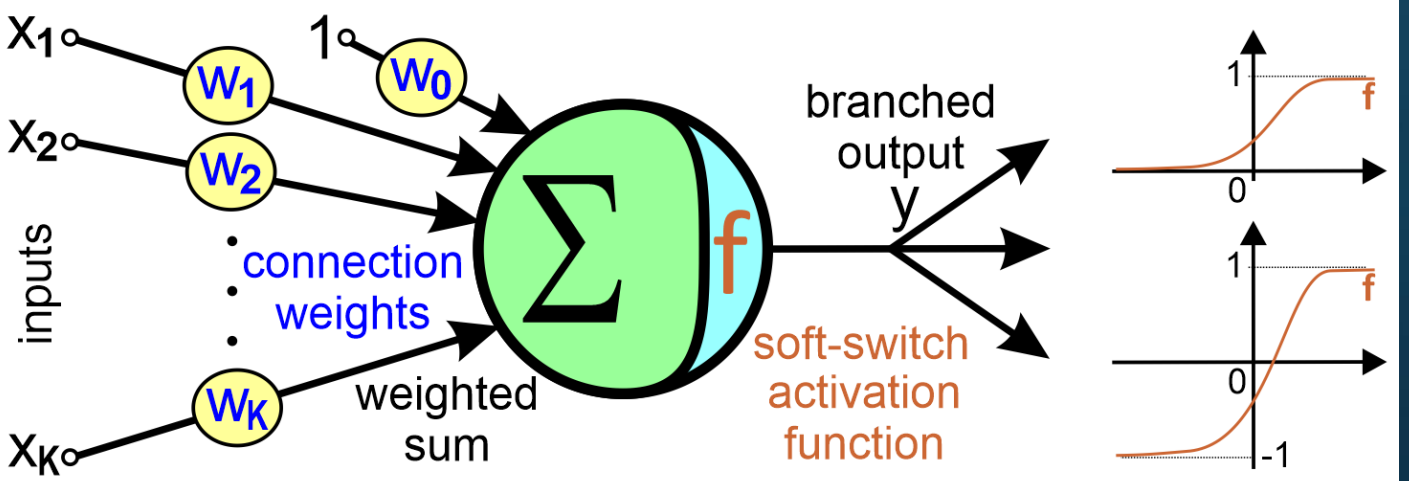
We shall try to implement them before implementing more advanced models and methods.

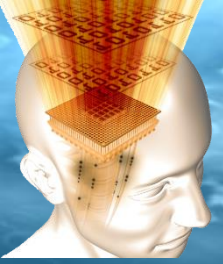
Use the soft-switch neurons with the sigmoidal activation function and construct an MLP network.

At first, we will try to use MLP networks for various classification tasks. It has a real practical value.

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







Structure of Multi-Layer Perceptron (MLP) Neural Network

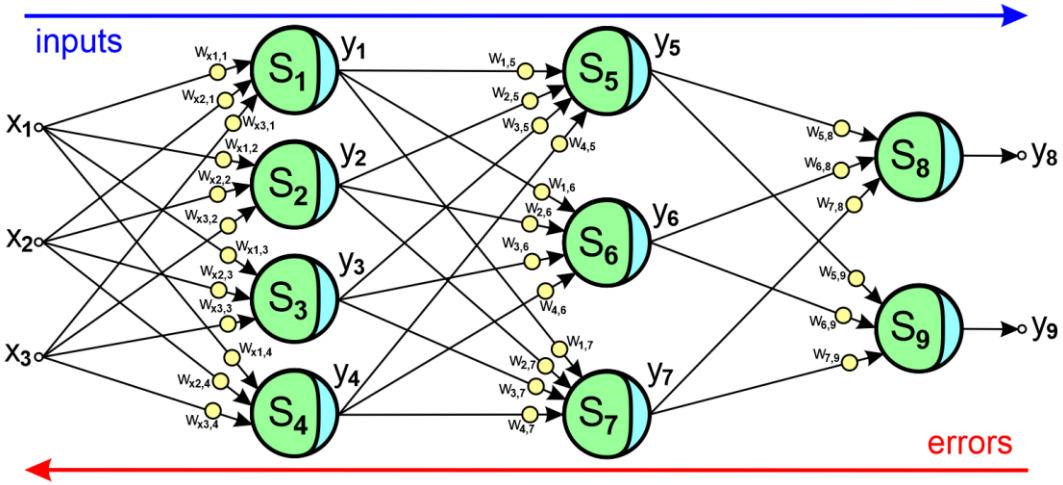
Use **matrices** and **vectors** to represent weights, neurons, and other parameters in the network.

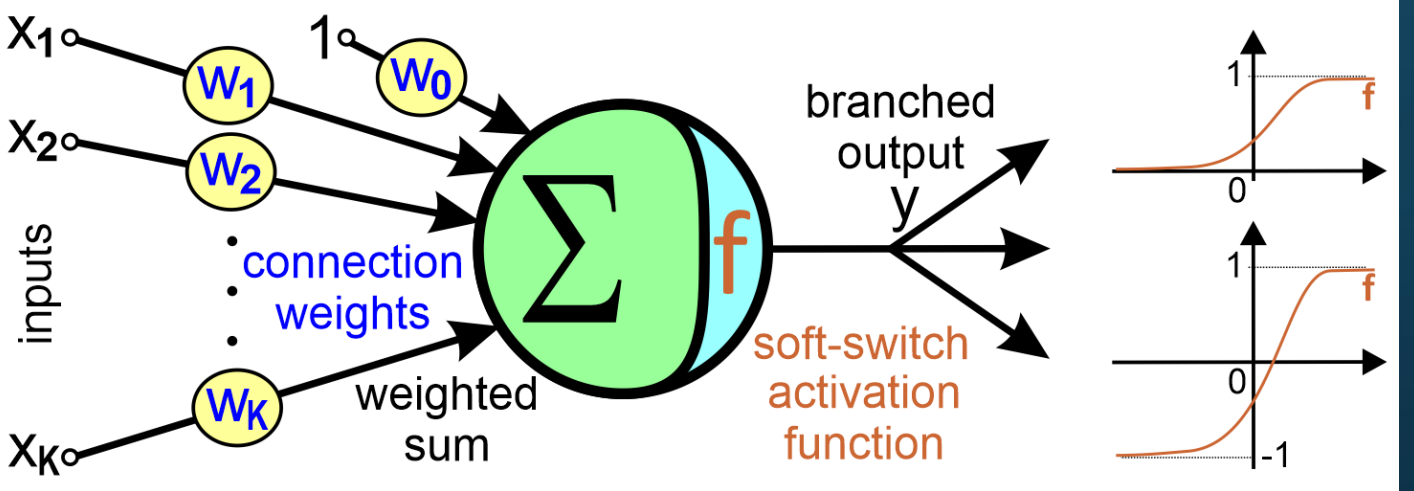
Weights of each layer can be represented by a single **matrix** which size is defined by the number of stimulating neurons (or nodes) [rows] and the number of the neurons which receives the stimuli [columns].

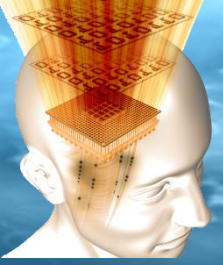
Neurons should be represented by **vectors** of classes which define weighted sums $\boldsymbol{S_n}$, outputs $\boldsymbol{y_n}$, errors $\boldsymbol{\delta_n}$ computed for these neurons during the backpropagation process.

Next, you multiply the input vectors $X_n = \{x_1, ..., x_N\}$ taken from the training data set $\{(X_1, d_1), ..., (X_N, d_N)\}$ with the matrix representing weights of the first layer hidden neurons, computing weighted sums and outputs.

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







Structure of Multi-Layer Perceptron (MLP) Neural Network

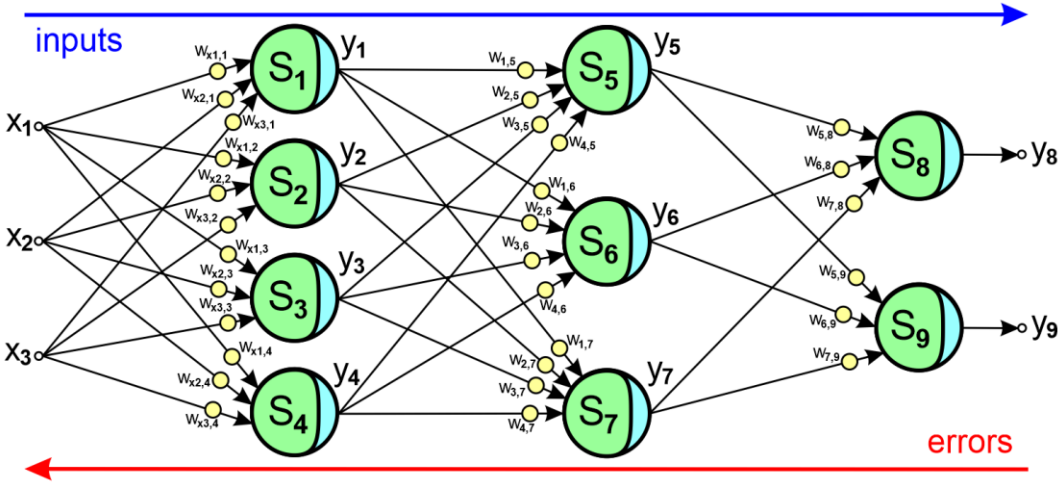
The output values computed inside the class vectors of neurons of the first hidden layer use to stimulate the neurons of the second hidden layer, etc.

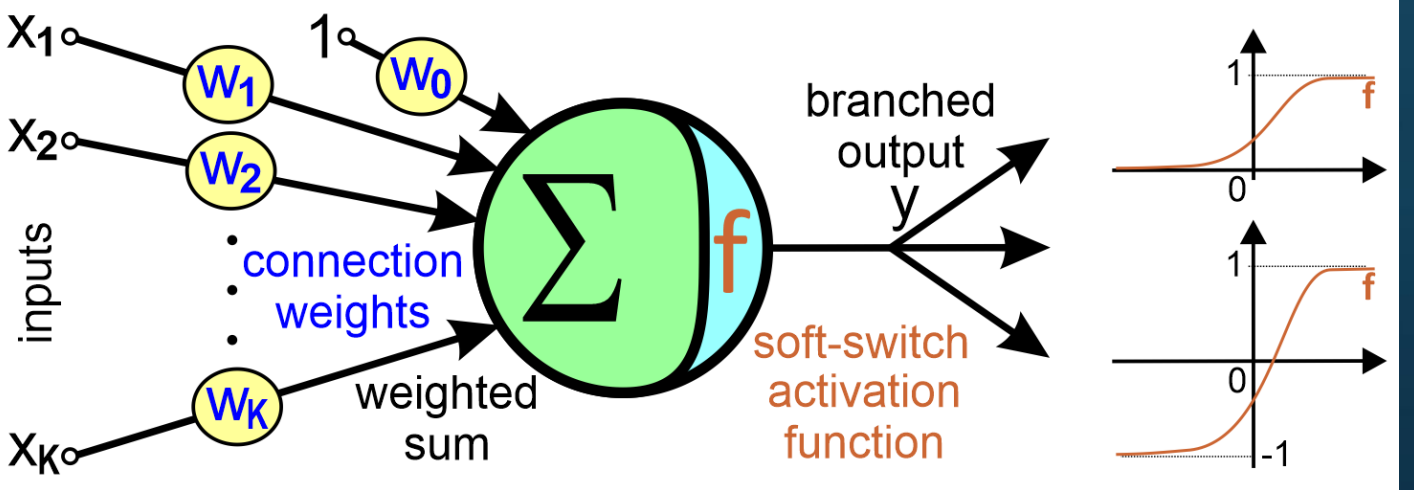
The propagation process is finished when you achieved the output neurons of the network.

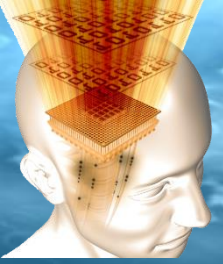
Next, compute the error vector $\boldsymbol{\delta_n} = \{\delta_1, \ldots, \delta_M\}$ on the basis of the desired output vector $\mathbf{d_n} = \{d_1, \ldots, d_N\}$ defined in the training data set for each input vector $\mathbf{X_n} = \{x_1, \ldots, x_N\}$.

Now, you can start propagating back the errors and updating weights. It is useful to use extra matrixes for representation of Δw.

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

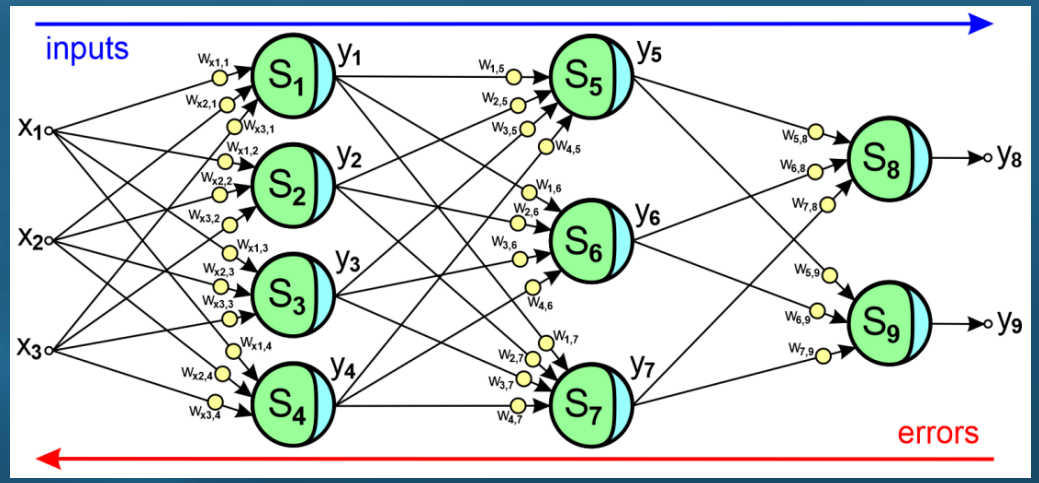






How to implement an MLP?

Each MLP neural network has a layer structure and is a kind of feedforward neural networks. This means that the stimulation of neurons of previous layer(s) can stimulate neurons of the next layer(s). Hence, no recurrent or reverse connections are possible!

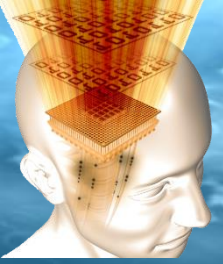


There are many ways of implementation of the MLP networks, but taking into account the fact that we will use them to create more complex deep architectures in the future, it is profitable to implement them in a universal way that will enable us to:

- add connections between various layers (not only the subsequent layers),
- add extra or remove some connections (i.e. not all-to-all neurons must be connected),
- connect various kinds of neurons (various activation functions) and various networks together,
- use various training routines for various layers of neurons in the future (when implementing deep approaches).

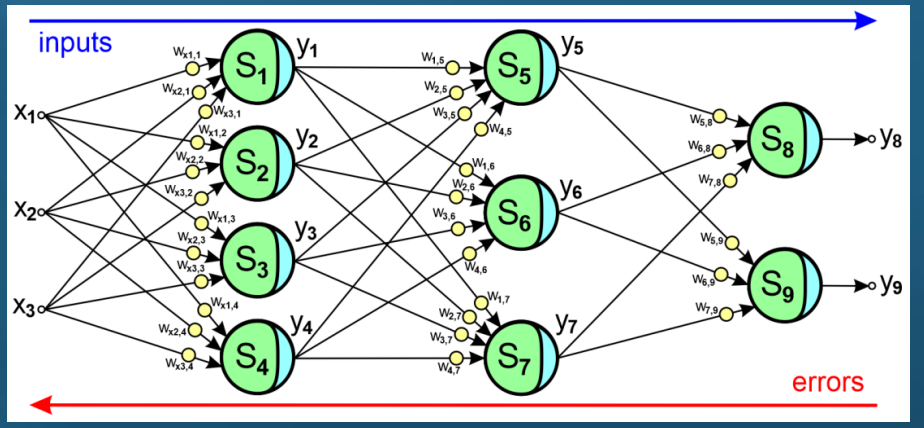
Hence, the neurons should be organized in layers processed in the same training step, but do not limit the number of layers, number of neurons in them, or possible connections between neurons of various layers.

Use objects and object-oriented programming to implement: neurons with dynamic lists of connections.



How to start implementation?

Construct an input interface to open data files containing training or testing data (text, spreadsheet, xml, database) and put them into a table (list) or several tables (lists).
Various types of attributes should be possible to be stored.
The attributes can be numerical (integer, float, date, time), symbolic (string) or boolean (bivalent {0,1} or fuzzy [0,1]).
The number of attributes should not be constant but limited.

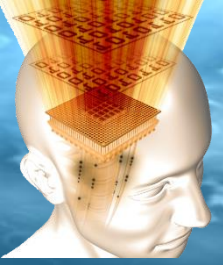


Attributes:	leaf-length	leaf-width	petal-length	petal-width	class
No	Numerical: float	Numerical: float	Numerical: float	Numerical: float	Symbolic: string
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
54	5.5	2.3	4	1.3	Iris-versicolor
55	6.5	2.8	4.6	1.5	Iris-versicolor
56	5.7	2.8	4.5	1.3	Iris-versicolor
57	6.3	3.3	4.7	1.6	Iris-versicolor
146	6.7	3	5.2	2.3	Iris-virginica
147	6.3	2.5	5	1.9	Iris-virginica
148	6.5	3	5.2	2	Iris-virginica
149	6.2	3.4	5.4	2.3	Iris-virginica
150	5.9	3	5.1	1.8	Iris-virginica

OR

No	sle	swi	ple	pwi	class name
1	5.1	3.5	1.4	0.2	SETOSA
2	4.9	3.0	1.4	0.2	SETOSA
3	4.7	3.2	1.3	0.2	SETOSA
4	4.6	3.1	1.5	0.2	SETOSA
5	5.0	3.6	1.4	0.2	SETOSA
54	5.5	2.3	4.0	1.3	VERSICOLOR
55	6.5	2.8	4.6	1.5	VERSICOLOR
56	5.7	2.8	4.5	1.3	VERSICOLOR
57	6.3	3.3	4.7	1.6	VERSICOLOR
146	6.7	3.0	5.2	2.3	VIRGINICA
147	6.3	2.5	5.0	1.9	VIRGINICA

The goal of these classes is to develop your CI system to process various data!



Desired Output Values Explanation

How do we define the output vectors d_n for training MLP networks?

For Iris data which define 3 classes (Setosa, Versicolor, and Virginica), we have to define output vectors consisting of 3 values:

1. for Iris Setosa
2. for Iris Versicolor
3. for Iris Virginica

FEATURE 1	FEATURE 2	FEATURE 3	FEATURE 4	CLASS
5,0	3,6	1,4	0,2	Iris-setosa
6,5	2,8	4,6	1,5	Iris-versicolor
6,5	3,0	5,2	2,0	Iris-virginica

				Setosa	Versicolor	Virginica
INPUT 1	INPUT 2	INPUT 3	INPUT 4	OUTPUT 1	OUTPUT 2	OUTPUT 3
5,0	3,6	1,4	0,2	1	0	0
6,5	2,8	4,6	1,5	0	1	0
6,5	3,0	5,2	2,0	0	0	1

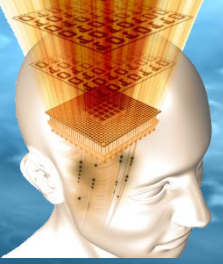
We always suppose that for the desired class, we should achieve 1 at the output of the neuron representing this class, and 0 for other classes, namely:

$d_i = [1 , 0 , 0]$ for Iris Setosa

$d_j = [0 , 1 , 0]$ for Iris Versicolor

$d_k = [0 , 0 , 1]$ for Iris Virginica

The winning neuron (the correct classification) is achieved when the neuron representing the desired class calculates the biggest value on its output in comparison to other outputs computed by other output neurons.



Implementation Tips and Tricks

Inputs should be implemented as a vector (or a matrix) of classes, from where neurons take input values to compute their internal weighted sums. These classes should contain subsequent input data.

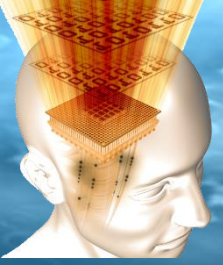
Each Neuron should be implemented as a class containing:

- a table/list of input connections (synapses) or pointers to connected neurons to the other neurons or the inputs, from where they take input values $x_1, \ldots, x_K$ to process and compute weighted sums S and the output values y,
- a table/list of output connections (synapses) or pointers to connected neurons to the other neurons or the outputs, from where they take calculated delta (error) parameters to calculate their delta parameters δ,
- a variable containing a weighted sum: S,
- a variable containing a delta parameter: δ,
- a variable containing an output value: y.
- When using the table/list of pointers to the connected neurons instead of the list of synapses, we need additionally to create a table of weights ($\mathbf{w}_k$) for input connections in each neuron and the same-size table of computed sums of updates of these weights ($\Delta\mathbf{w}_k$) when using batch training. On the other hand, when using the table/list of synapses, the weight is stored inside the class implementing a synapsis.

We could do it much easier using tables of neurons and tables of pointers to the connected neurons or using matrices, but the above-presented implementation model will let us change the connection lists easier to use them in deep learning algorithms, networks, and strategies, so this kind of implementation is simply recommended, however, fill free to do it as you like if you have any other idea how to implement it efficiently and satisfy our contemporary and future goals.

Neurons should be organized in layers. The layer class consists of a table/list of neurons.
The entire DeepMLP network (or each MLP subnetwork) should consist of a list of subsequent layers.

Outputs should be also implemented as a vector or a matrix of classes, which will contain output values of a neural network, desired outputs (taken from training data), and compute delta parameters to propagate them back.



How to represent weights and connections?

Generally, we can represent weights (w_x) and input connections (In_x) it three different ways:

1. Using tables (inside each neuron) representing vectors of indices (I_x) pointing out connected neurons and in the same way, we represent weights using tables of weight values:

In1	In2	In3	In4	In5	In6	In7	In8	In9
I1	I2	I3	I4	I5	I6	I7	I8	I9

Indices of nine input connections inside each neuron

In1	In2	In3	In4	In5	In6	In7	In8	In9
w1	w2	w3	w4	w5	w6	w7	w8	w9

Weight vector of nine input connections inside each neuron

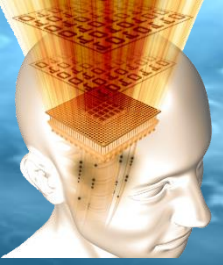
2. Using matrices (inside each layer of neurons) representing vectors of indices in rows for each neuron in the layer, pointing out connected neurons and in the same way, we represent weights using matrices of weight values for each layer of neurons:

	In1	In2	In3	In4	In5	In6	In7	In8	In9
N1	I11	I12	I13	I14	I15	I16	I17	I18	I19
N2	I21	I22	I23	I24	I25	I26	I27	I28	I29
N3	I31	I32	I33	I34	I35	I36	I37	I38	I39
N4	I41	I42	I43	I44	I45	I46	I47	I48	I49
N5	I51	I52	I53	I54	I55	I56	I57	I58	I59

Indices of nine input connections for the layer consisting of five neurons

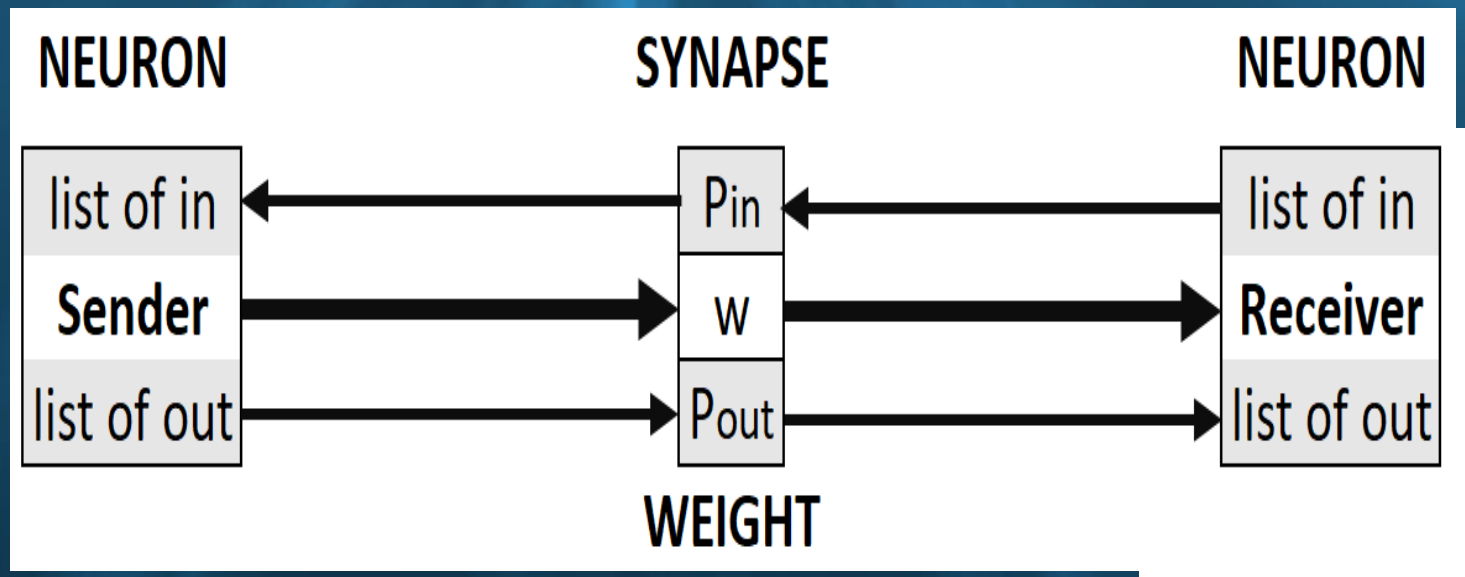
	In1	In2	In3	In4	In5	In6	In7	In8	In9
N1	w11	w12	w13	w14	w15	w16	w17	w18	w19
N2	w21	w22	w23	w24	w25	w26	w27	w28	w29
N3	w31	w32	w33	w34	w35	w36	w37	w38	w39
N4	w41	w42	w43	w44	w45	w46	w47	w48	w49
N5	w51	w52	w53	w54	w55	w56	w57	w58	w59

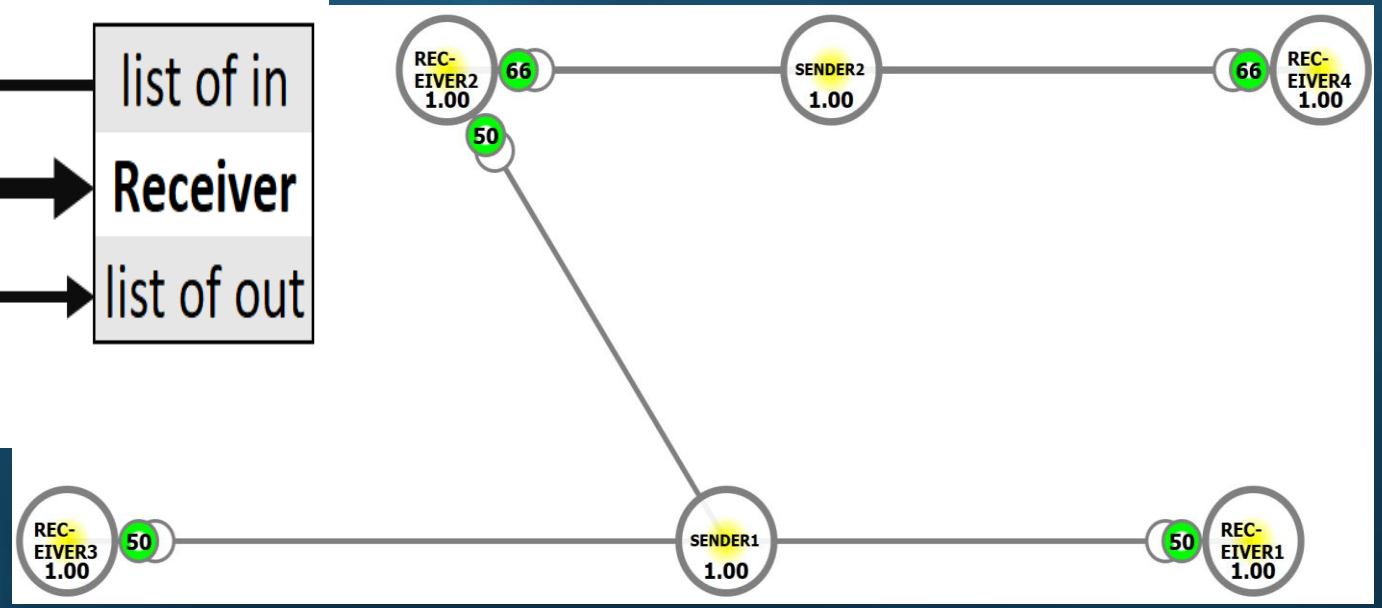
Weight martix of input connections for the layer consisting of five neurons

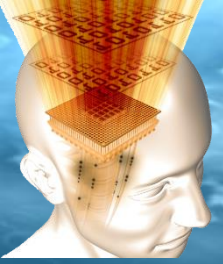


How to represent weights and connections?

3. Using list/tables of pointers to the instances of the class representing synapses which contain weights and pointers to the input and output neuron of this synaptic connection, while neurons have two lists of synapses: the first one for input connections (the list of input synapses), and the second one for output connections (the list of output synapses):







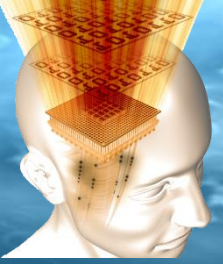
Input Data Preprocessing

It is usually beneficial to normalize input data when using the backpropagation algorithm and the sigmoidal activation function:

$$y_i = \frac{x_i - x_i^{min}}{x_i^{max} - x_i^{min}}$$

- $x = [x_1, x_2, \ldots, x_N]$ – is the vector of raw input data,
- $y = [y_1, y_2, \ldots, y_N]$ – is the vector of normalized input data.
- x_i^{min} – is the minimum value of the i-th value (attribute)
- x_i^{max} – is the maximum value of the i-th value (attribute)

Unfortunately, normalization is sensitive to outliers and scattered data!



Initializing the Neuronal Structure

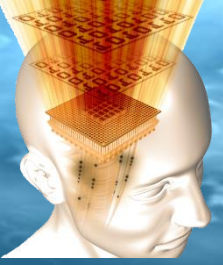
It is necessary to construct the neural network structure when using the backpropagation algorithm because this method has not built-in any procedure for reconstructing or developing it.

We can just try to guess the suitable topology or try to use genetic algorithms or evolutionary approaches to this task.

If you try to guess the network topology for a given dataset, start with a small number of neurons in a single hidden layer. If it is not enough and the results of training are not satisfying, try to add several neurons to this layer or add an extra hidden layer. Usually, subsequent hidden layers have fewer numbers of neurons than the previous ones.

If the constructed neural network should be able to generalize training data well we have to build the structure in such a way that the network has much fewer weights (N_w) than the number of trained data (N) taking into account also the number of data attributes (N_A), i.e. $N_w << N \cdot N_A$.
On the other hand, the network can try to learn all training data too precisely,
but it will generalize poorly (this phenomenon is called overlearning).

Furthermore, the weights of the created structure must be initiated in small numbers, usually say not bigger than [-5, 5], but the initializing range is often even smaller, e.g. [-0.1, 0.1].



Delta Rule for Neuron Adaptation

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

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

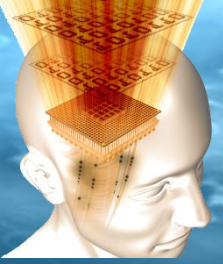
The correction of the weight for differentiable activation function *f* is computed after:

$$\Delta \mathrm{w}_k = \eta \cdot \delta_n \cdot f'(S) \cdot \mathrm{x}_k \qquad where \quad \delta_n = d_n - y_n$$

where *f'* is the differential of function *f*.

When the activation function is sigmoidal then we achieve the following expression:

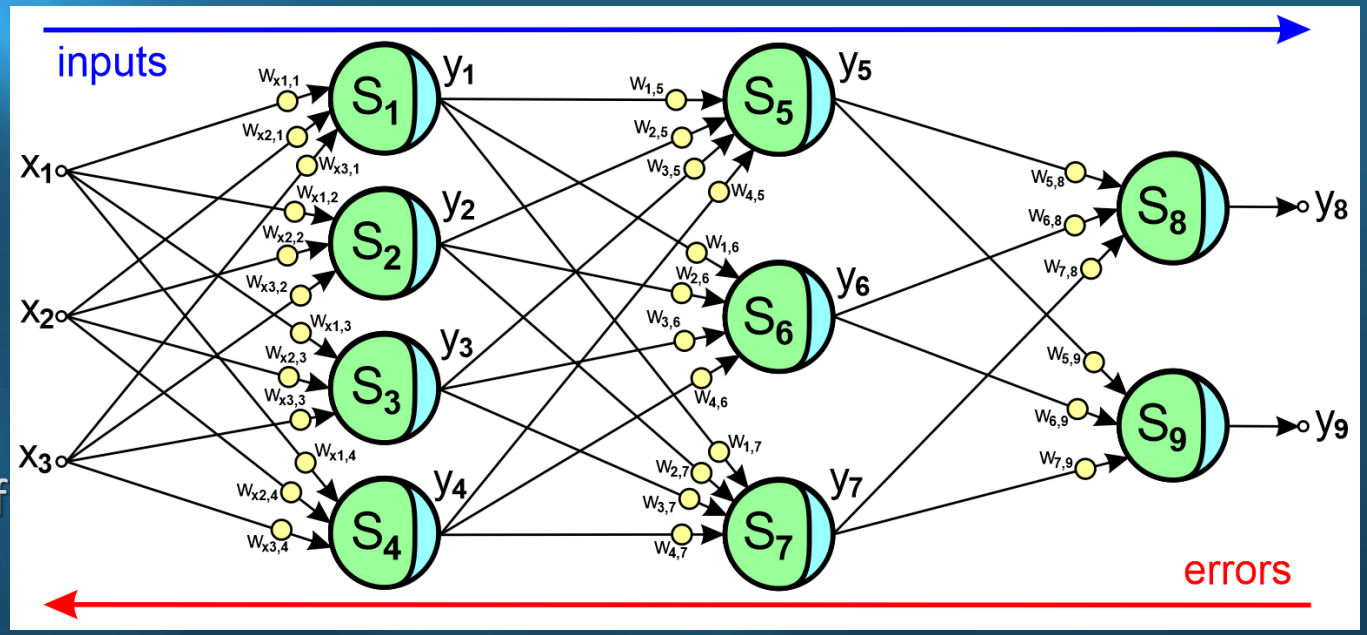
$$\Delta \mathrm{w}_k = \eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot \mathrm{x}_k \qquad where \quad \delta_n = d_n - y_n$$



Backpropagation Algorithm

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

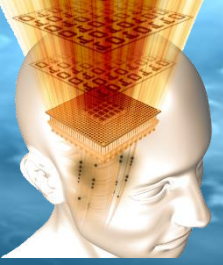
1. The input propagation phase propagates the inputs throughout all hidden layers to the output layer neurons. In this phase, neurons make the summation of weighted inputs taken from the neurons in the previous layer.
2. The error propagation phase propagates back the errors (delta values) computed on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

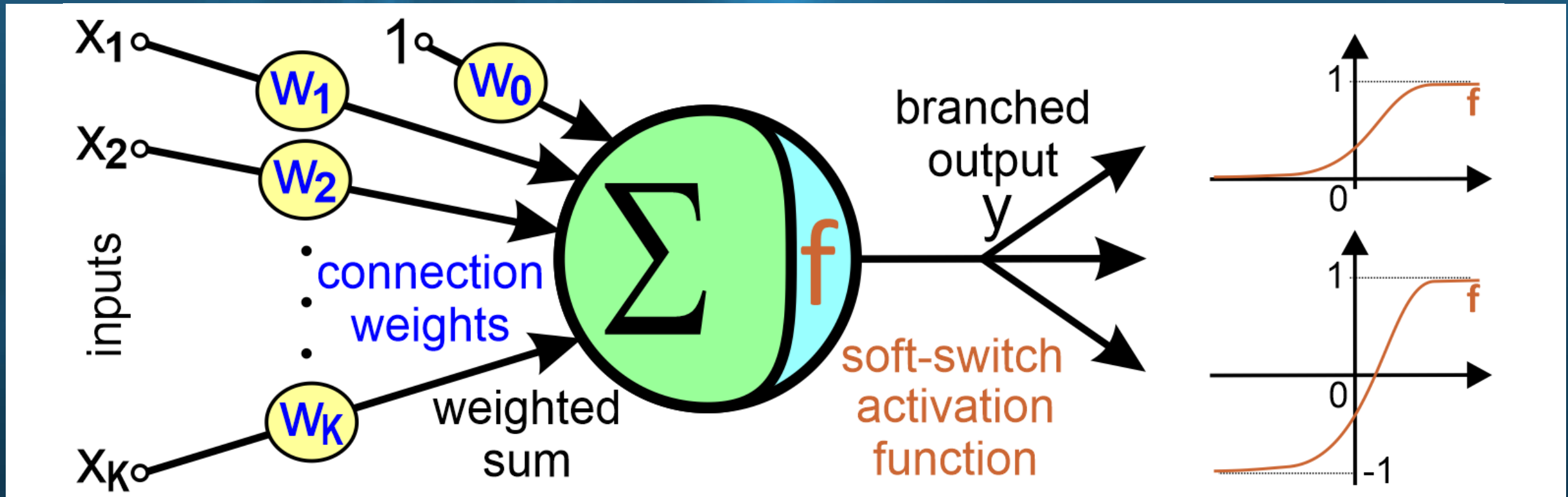
- the computed corrections immediately after their computation during the online training,
- the average value of all computed corrections of each weight after finishing the whole training cycle for all training samples during the offline (batch) training.

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

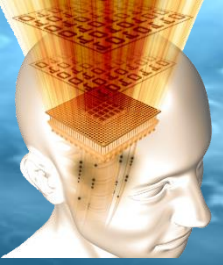


Soft-Switch Perceptron

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:



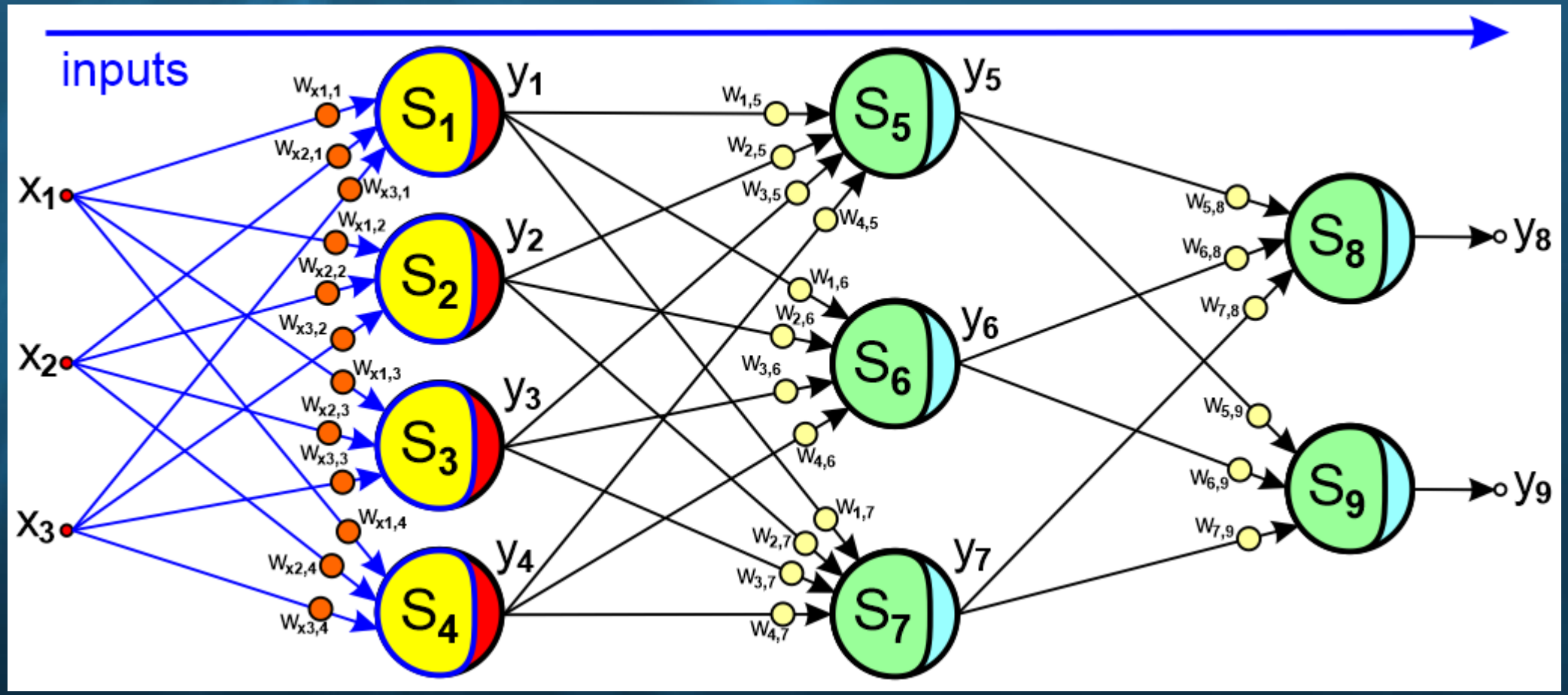
$$y = f(S) = \frac{1}{1 + e^{-\beta \cdot S}} \in (0, 1) \quad OR \quad y = f(S) = \frac{2}{1 + e^{-\beta \cdot S}} - 1 \in (-1, 1) \quad OR \quad y = f(S) = \tanh(\beta \cdot S) \in (-1, 1)$$

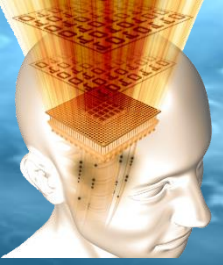


Backpropagation Algorithm

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:

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

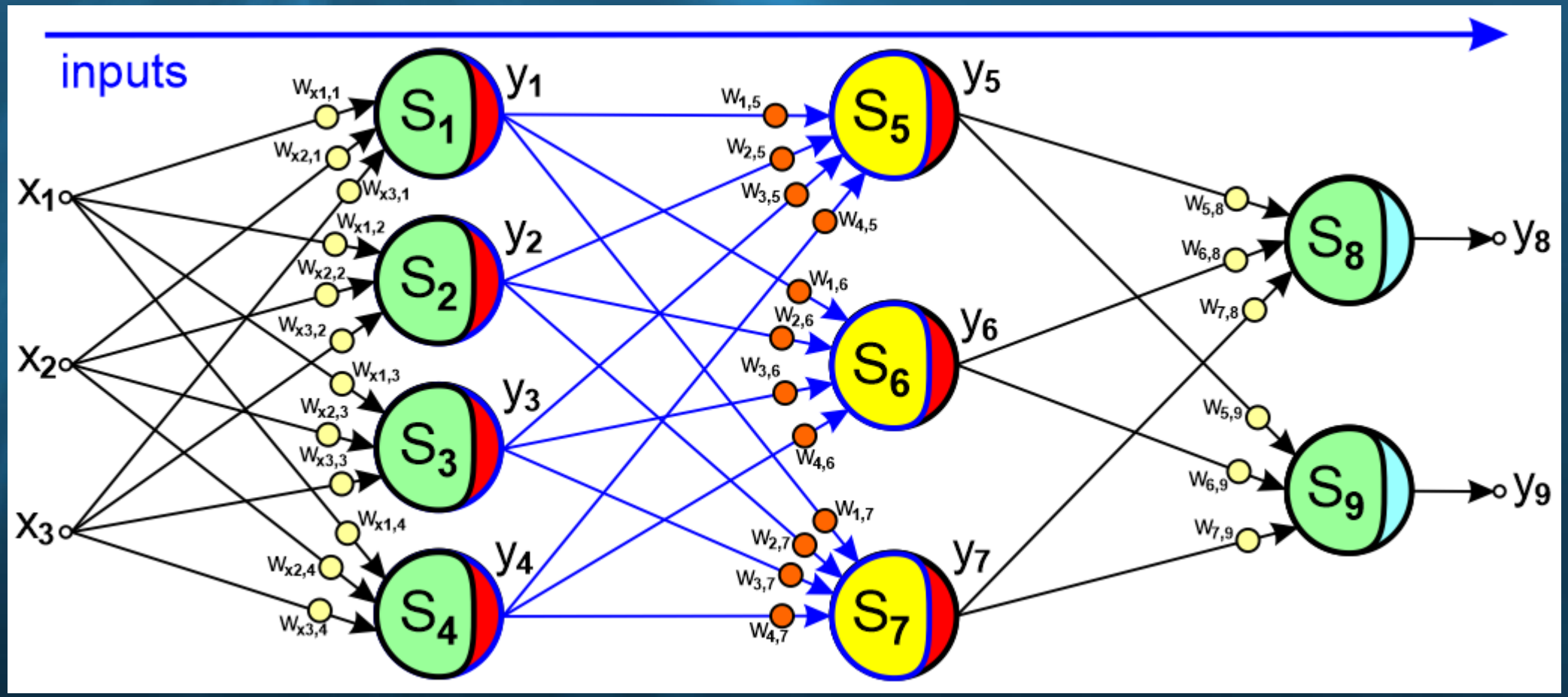


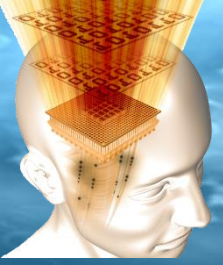


Backpropagation Algorithm

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:

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

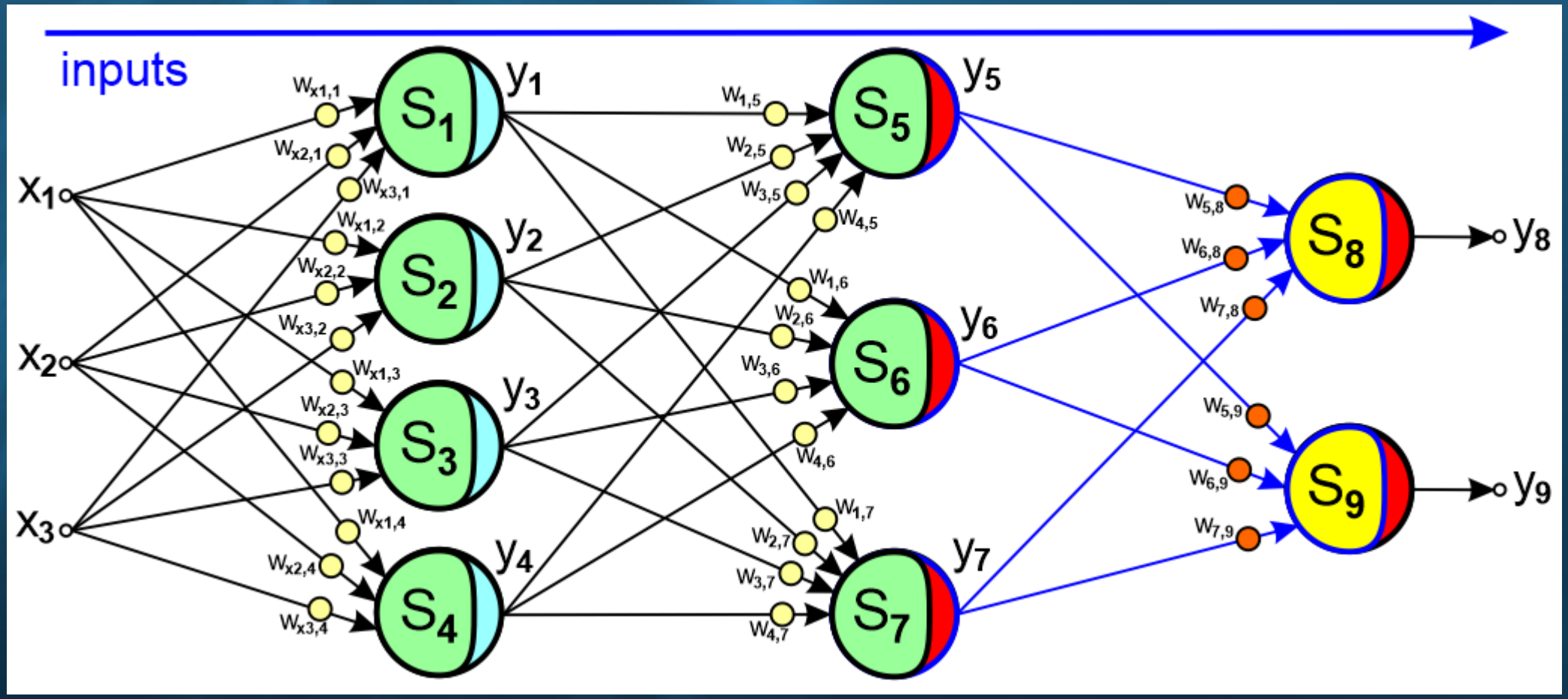


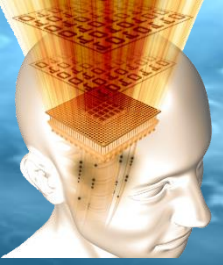


Backpropagation Algorithm

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

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

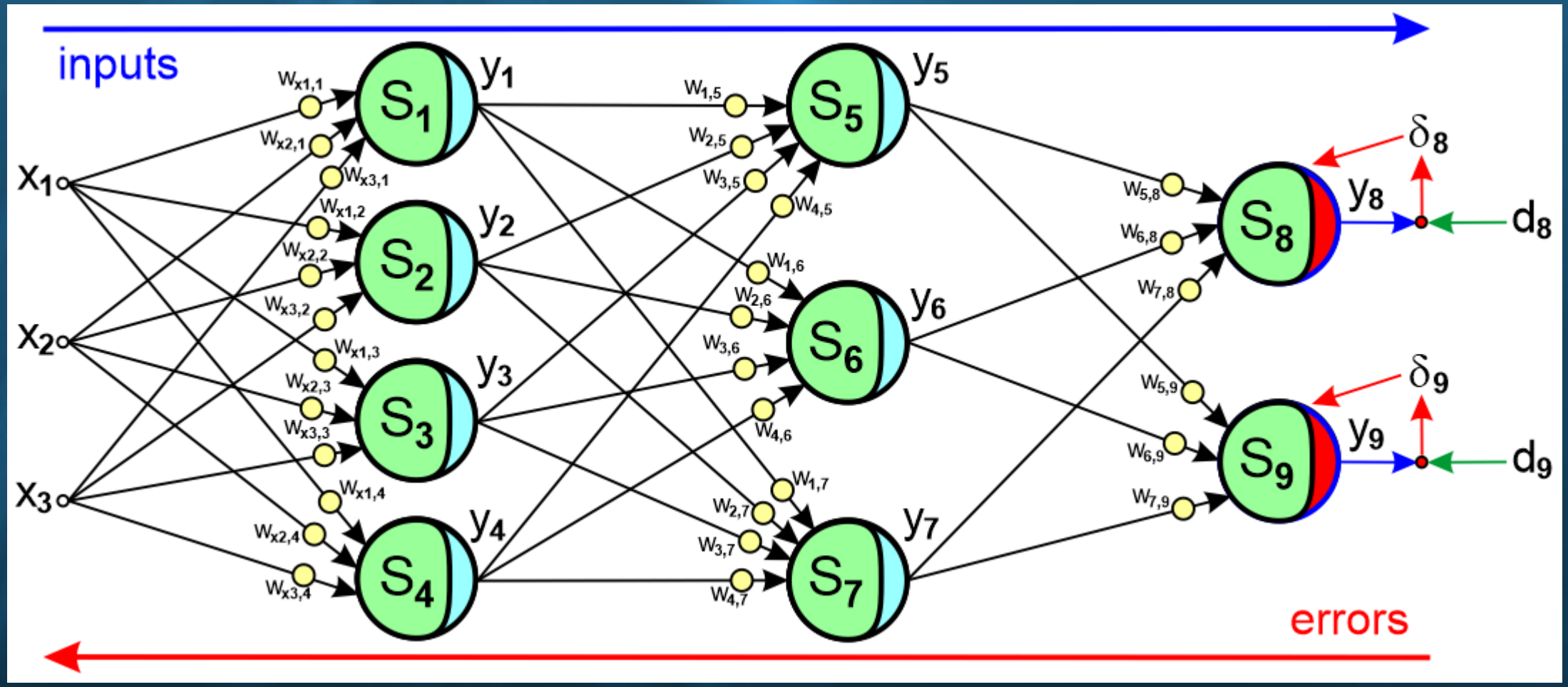


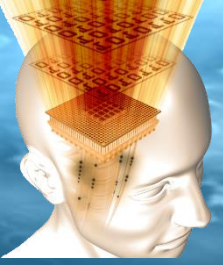


Backpropagation Algorithm

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.

$$\delta_n = d_n - y_n$$



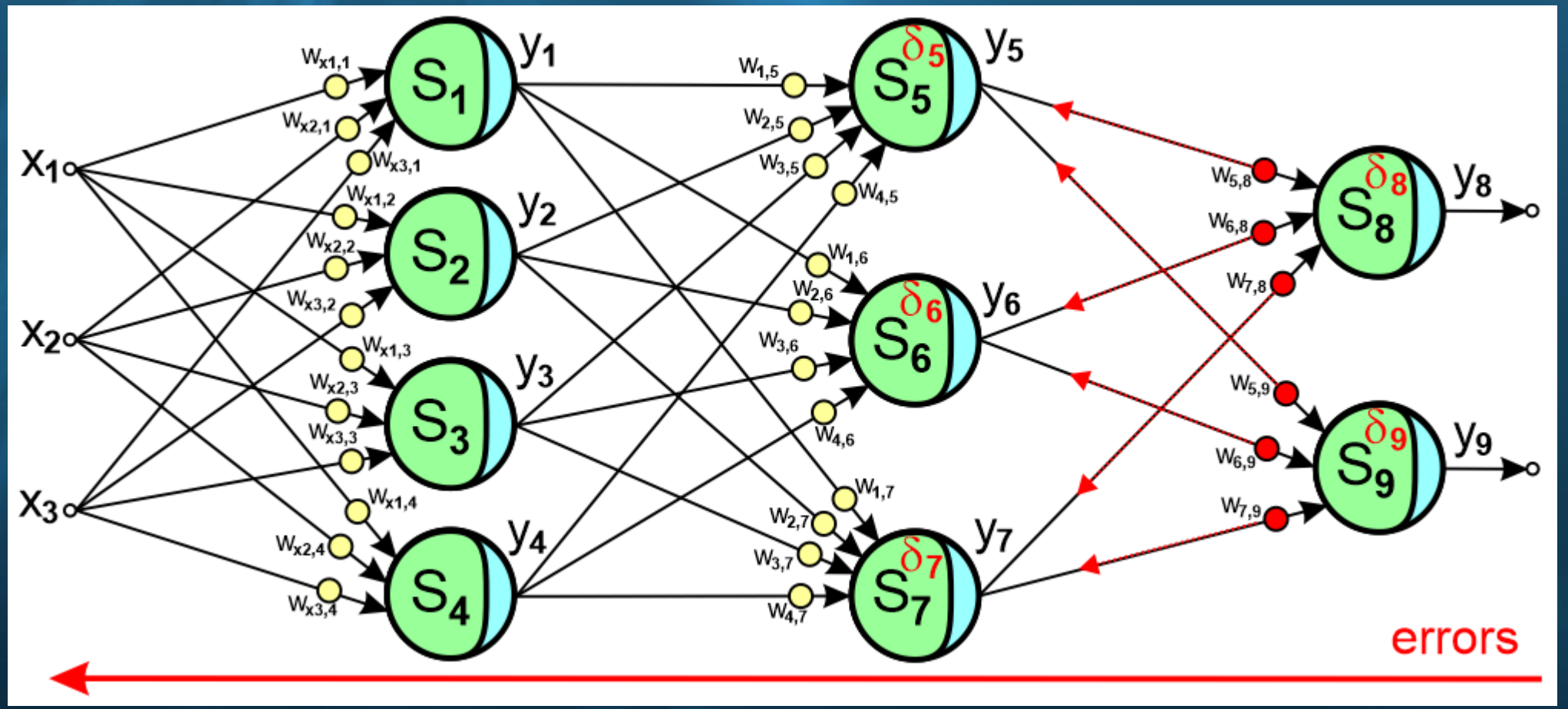


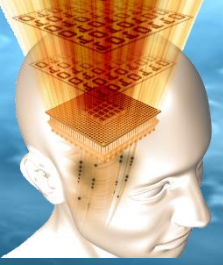
Backpropagation Algorithm

The errors δ_8, 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:

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

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



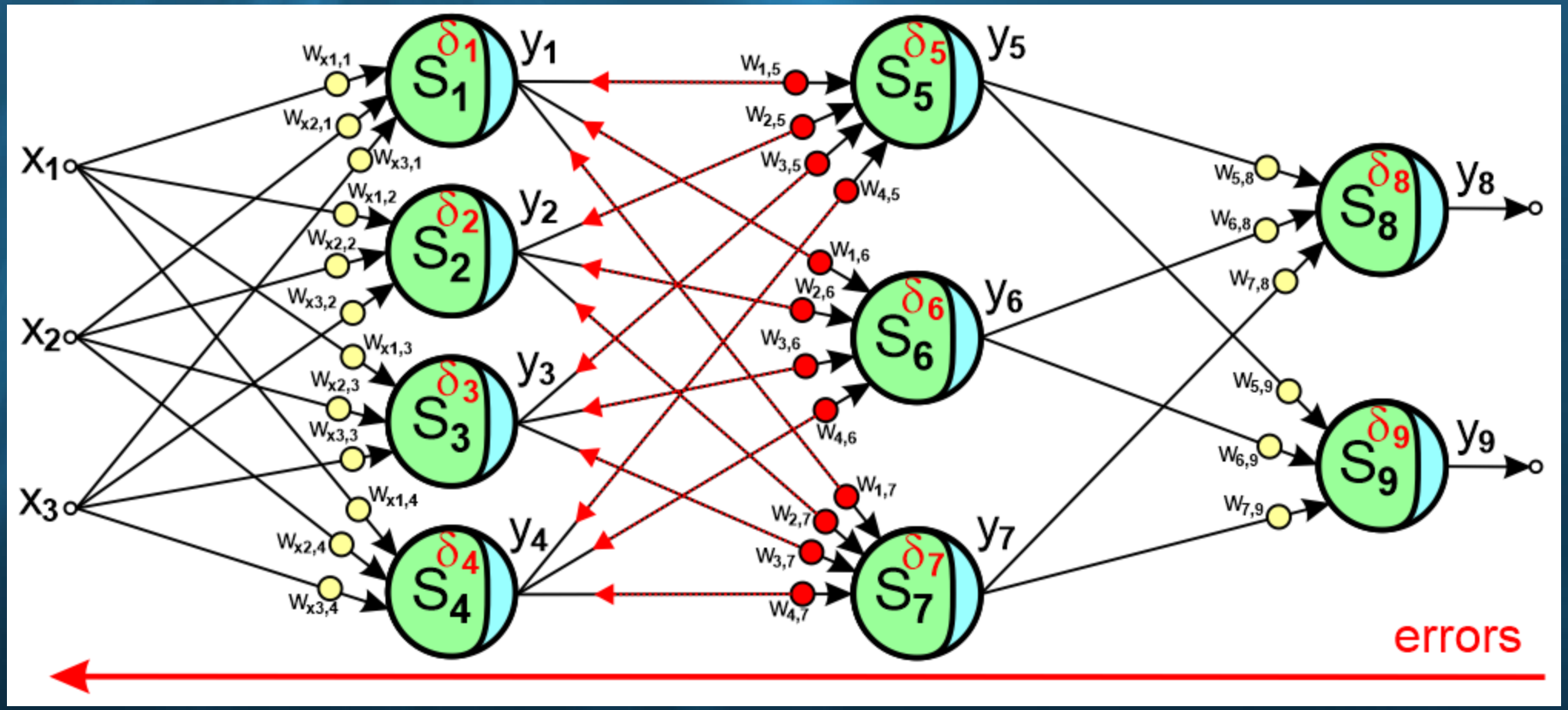


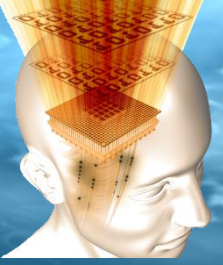
Backpropagation Algorithm

Next, the errors δ_5, δ_6, 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:

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

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

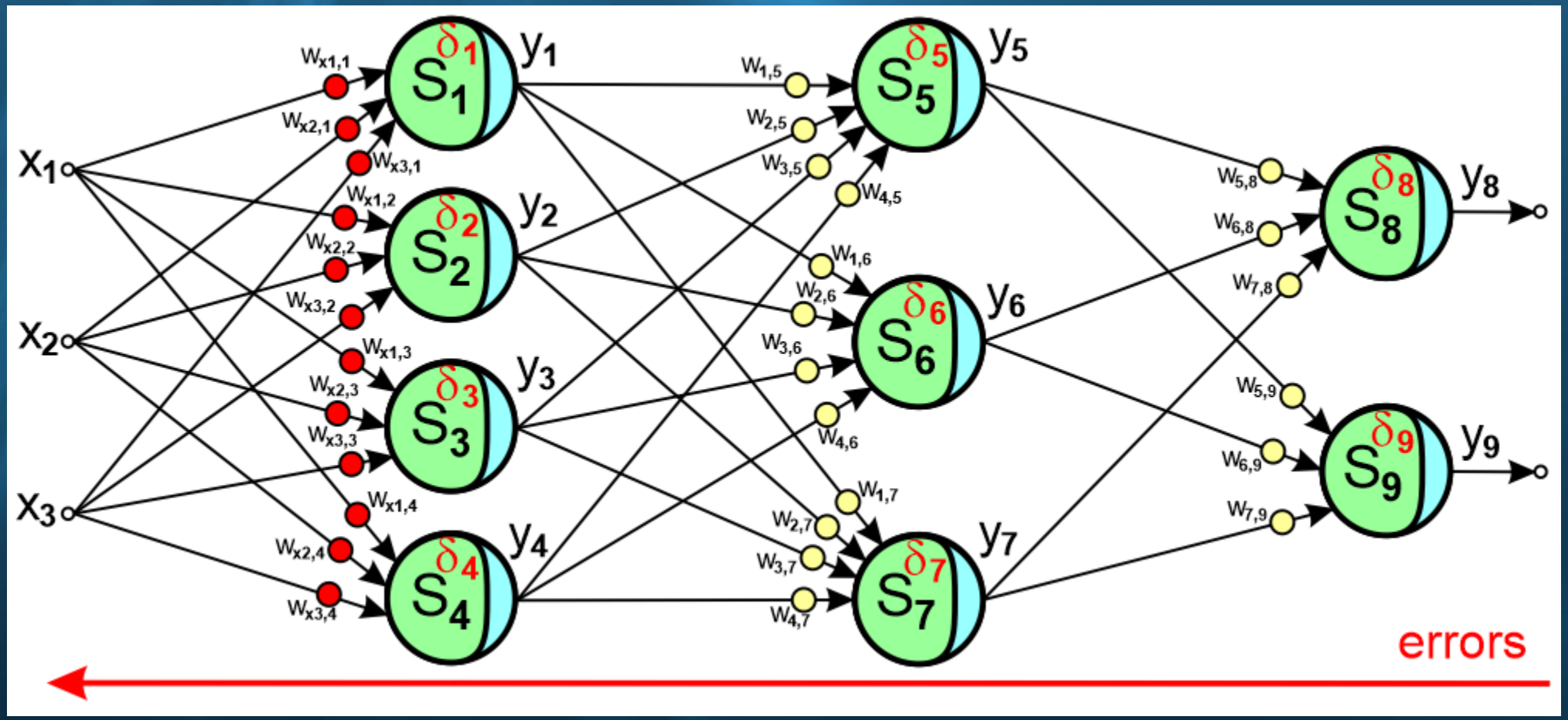


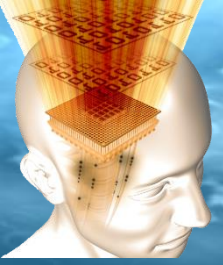


Backpropagation Algorithm

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:

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





Initialization & Training Parameters

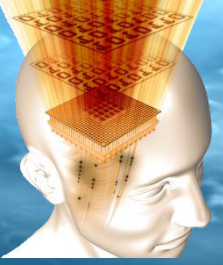
The number of hidden layer neurons should be higher rather than lower.
However, for easy problems, one or two hidden layers may suffice.

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 and implemented later.

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 -5 to +5, or even smaller.

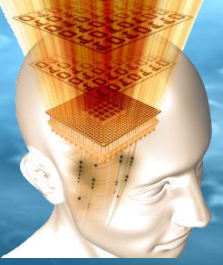
The learning rate η should be adjusted stepwise ($\eta < 1$), considering stability requirements. However, since convergence is usually rather fast when the error becomes very small, it is advisable to reinstate η to its initial value before proceeding.

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.



Experiments with Training Data

1. First, use some easy and small training data, like Iris data or Wine data.
2. Next, experiment with more difficult ones (downloaded from ML Repository) and try to overcome training difficulties that can occur.
3. Use cross-validation that will be described and discussed later during the lectures.
4. Use a various number of layers and neurons.
5. Use genetic and evolutional approaches to find out possibly the best network topology and initial weights after these methods will be presented during the lectures.
6. Use various or adjusted learning rate η during the training process.
7. Try 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.
8. Use deep learning strategies to achieve better results of training.



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 ($0 < \alpha < 1$), but it may not always work or could harm convergence:

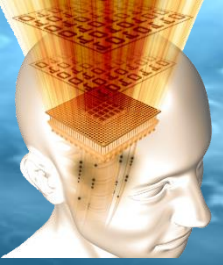
$$\Delta \mathbf{w}_{k,n}^{p} = \alpha \cdot \Delta \mathbf{w}_{k,n}^{p-1} + \eta \cdot \delta_n \cdot f'\left(\sum_{k=0}^{K} \mathbf{x}_k \cdot \mathbf{w}_k\right) \cdot \mathbf{x}_k = \alpha \cdot \Delta \mathbf{w}_{k,n}^{p-1} + \eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot \mathbf{x}_k$$

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

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

$$= \alpha \cdot \Delta \mathbf{w}_{k,n}^{p-1} + (1 - \alpha) \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot \mathbf{x}_k$$

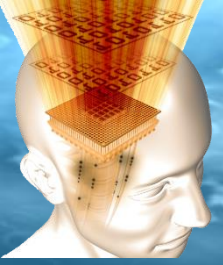
where *p* is the training period (cycle) of training samples.



Overcome Convergence Problems

In order to overcome convergence problems of the backpropagation algorithm we can:

- Change the range of the sigmoid function from [0, 1] to [-1, 1].
- Modifying step size (learning rate η) during the adaptation process.
- Start 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 than the casual ones.
- Reduce the number of inputs to overcome the curse of dimensionality problem.
- Use deep learning strategies and networks.
- Use cross-validation to avoid the problem of over-fitting.

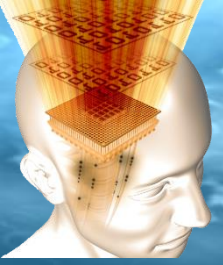


Use K-fold Cross-Validation to overcome convergence problems

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



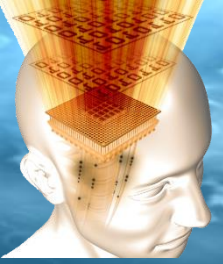
K-fold Cross-Validation

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 better validate the model 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 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



K-fold Cross-Validation

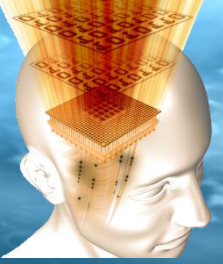
The way of selection of the test patterns in each training step should be representative and proportional from each class point of view regardless of the cardinality of classes!
We have to consider how the training data are organized in the training dataset:

- 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



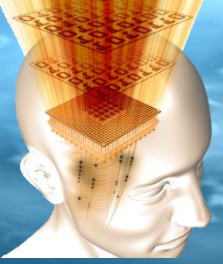
K-fold Cross-Validation

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	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	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	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	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	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	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

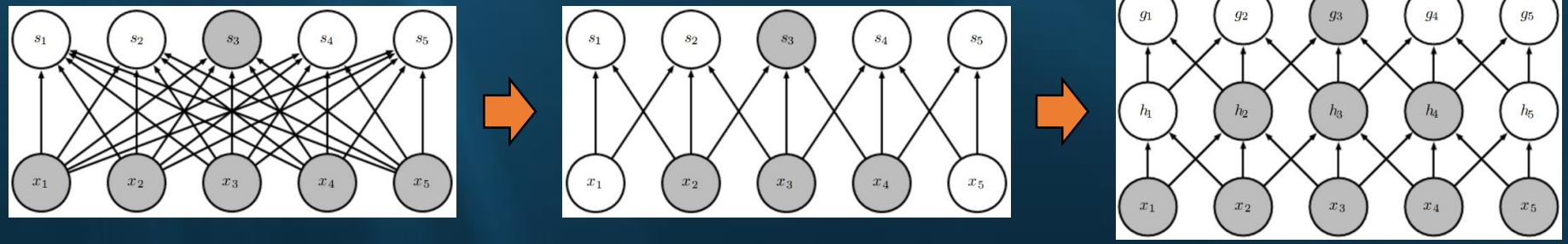
The choice between various options should be made on the basis of the initial order or disorder of patterns of all classes in the dataset to achieve representative selection of the test patterns used for the validated model.

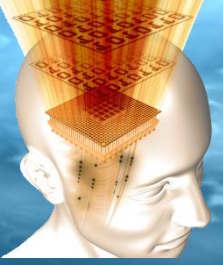


Try to Use a Simple Deep Learning Strategy

Try to use a simple deep learning strategy to upgrade your MLP Network:

- using extra layers added gradually, additional neurons,
- update only a selected part of neurons 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 of 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.





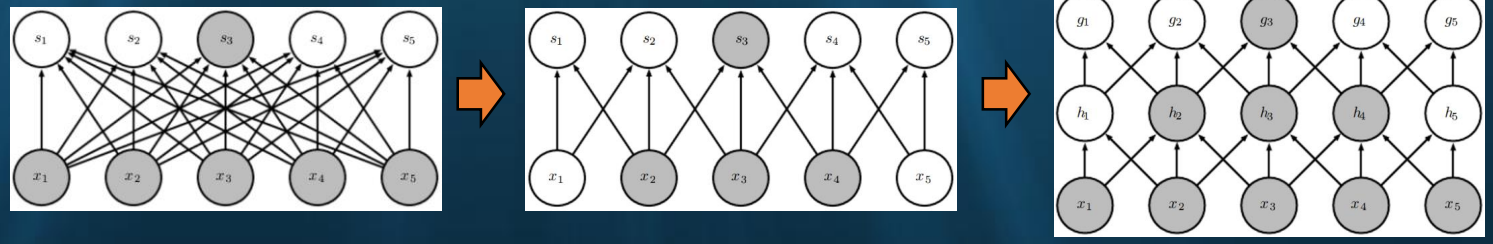
Try to Use a Simple Deep Learning Strategy

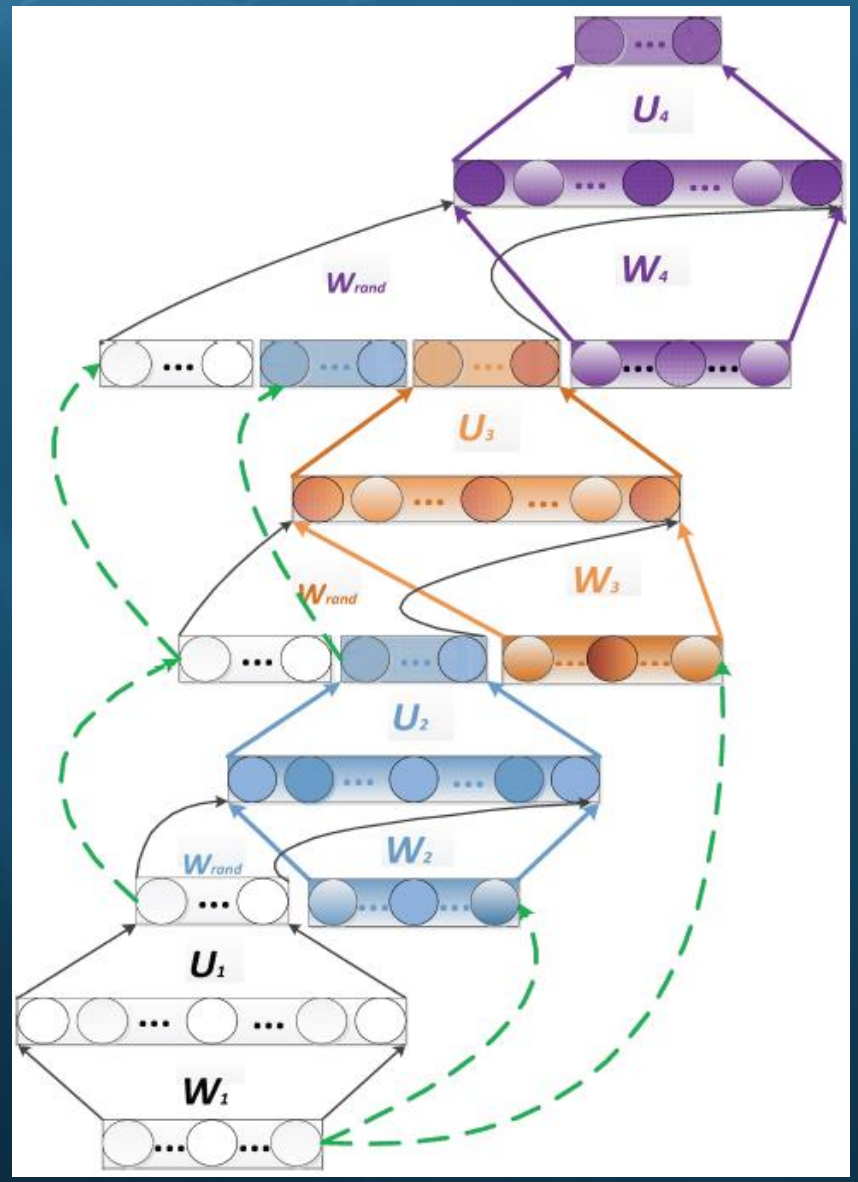
Use neurons that have input connections coming from different layers, combining the variety of the previously extracted features to compute their outputs.

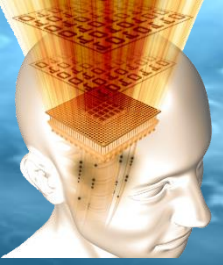
During our laboratory classes:

Try to use this strategy instead of the classic MLP all-to-all connections and compare achieved training results.

Use it together with limited number of connections between neurons in the successive layers.

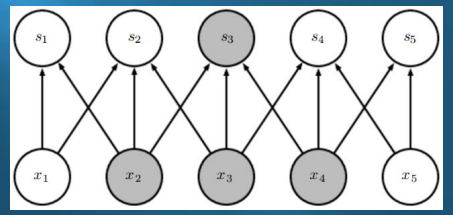




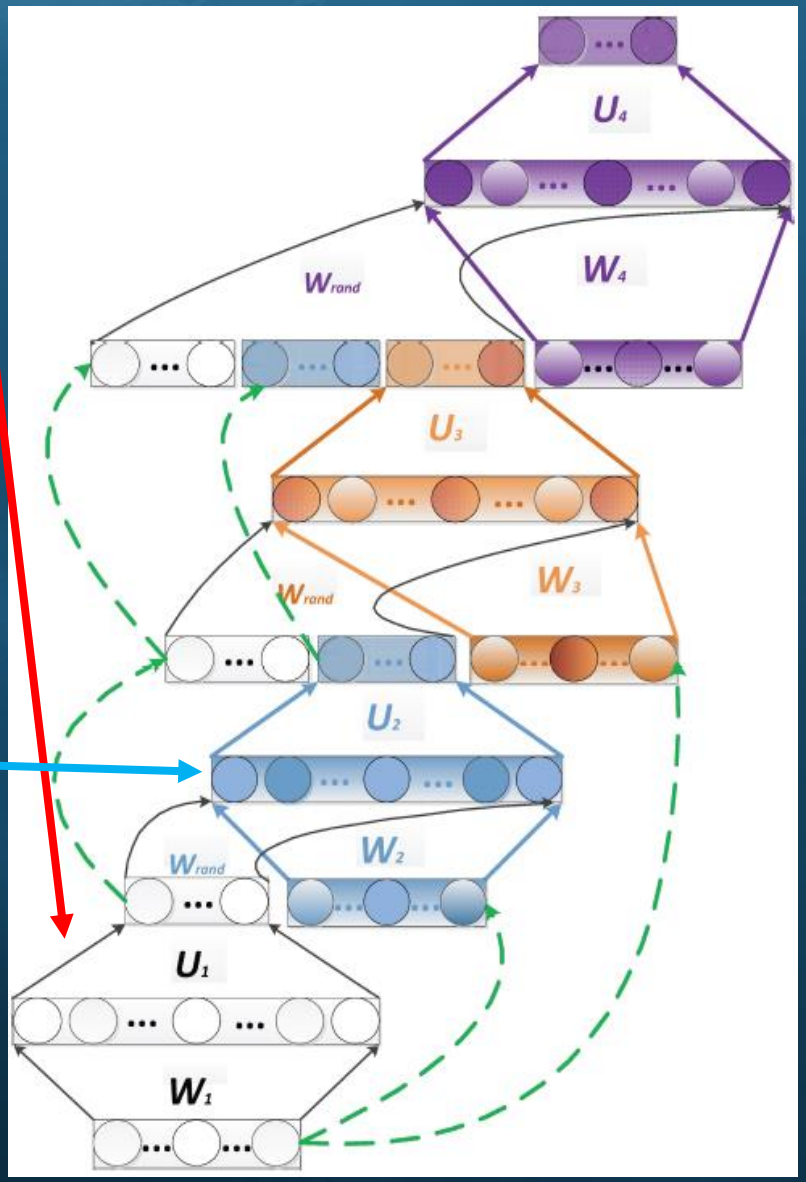


Construct Deep MLP Structure

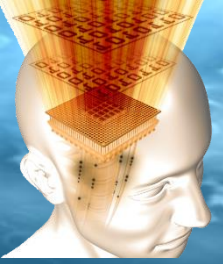
1. Create a simple MLP network (e.g. for Iris data consisting of 4 inputs, 3 outputs and a single hidden layer consisting of a few neurons (5 - 20) to achieve possibly good generalization properties. You can also use sparse aggregated connections:

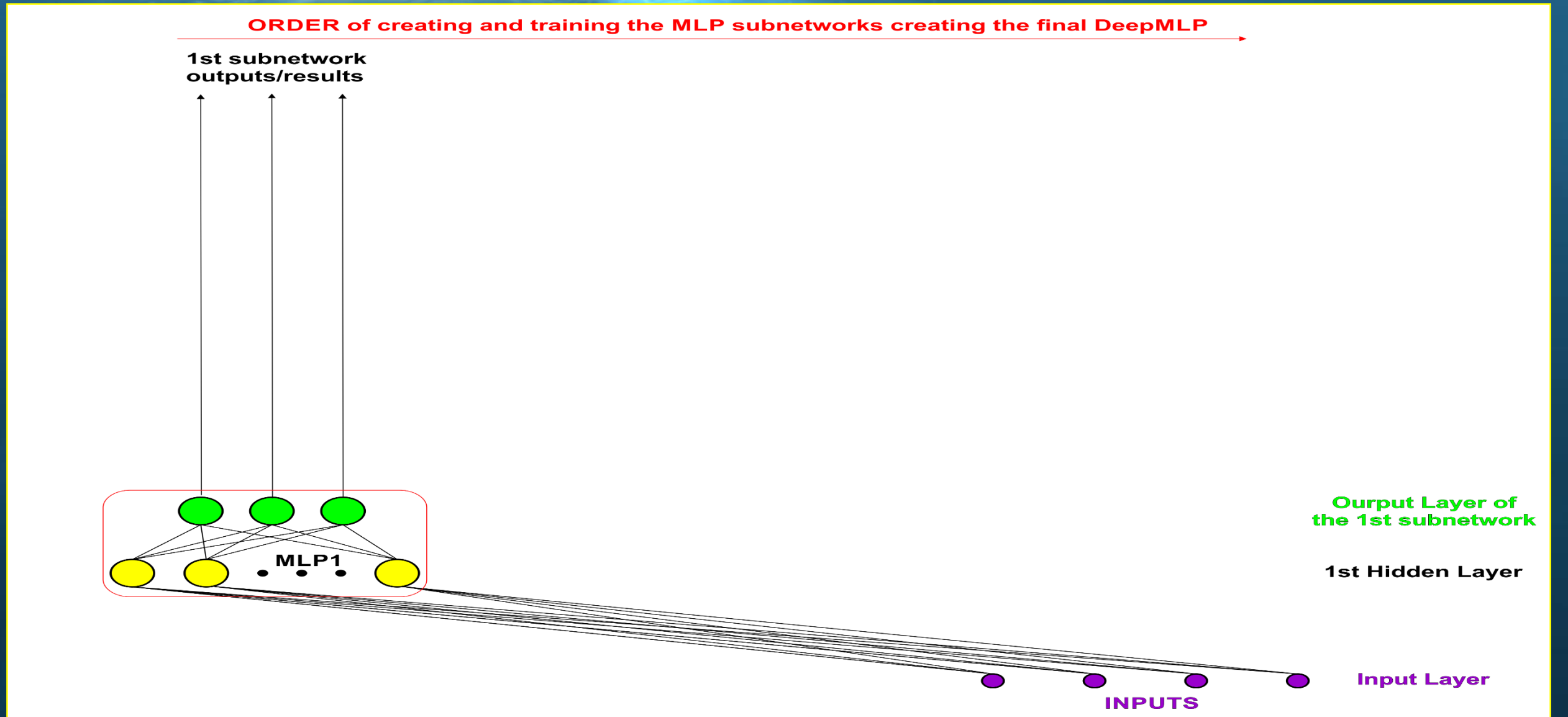


2. Learn this network using backpropagation algorithm in a number of steps (50 – 500) + (later) cross-validation until the network error is not lower than a given training level.
3. Next, add a next subnetwork with a next hidden layer, and learn this network with/without (two options) changing the weights of the previous subnetwork created in step 1. Try to achieve better results than for the first network, training it until the results are not significantly better. Continue such a process until you will get satisfactory training results.

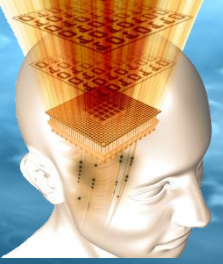


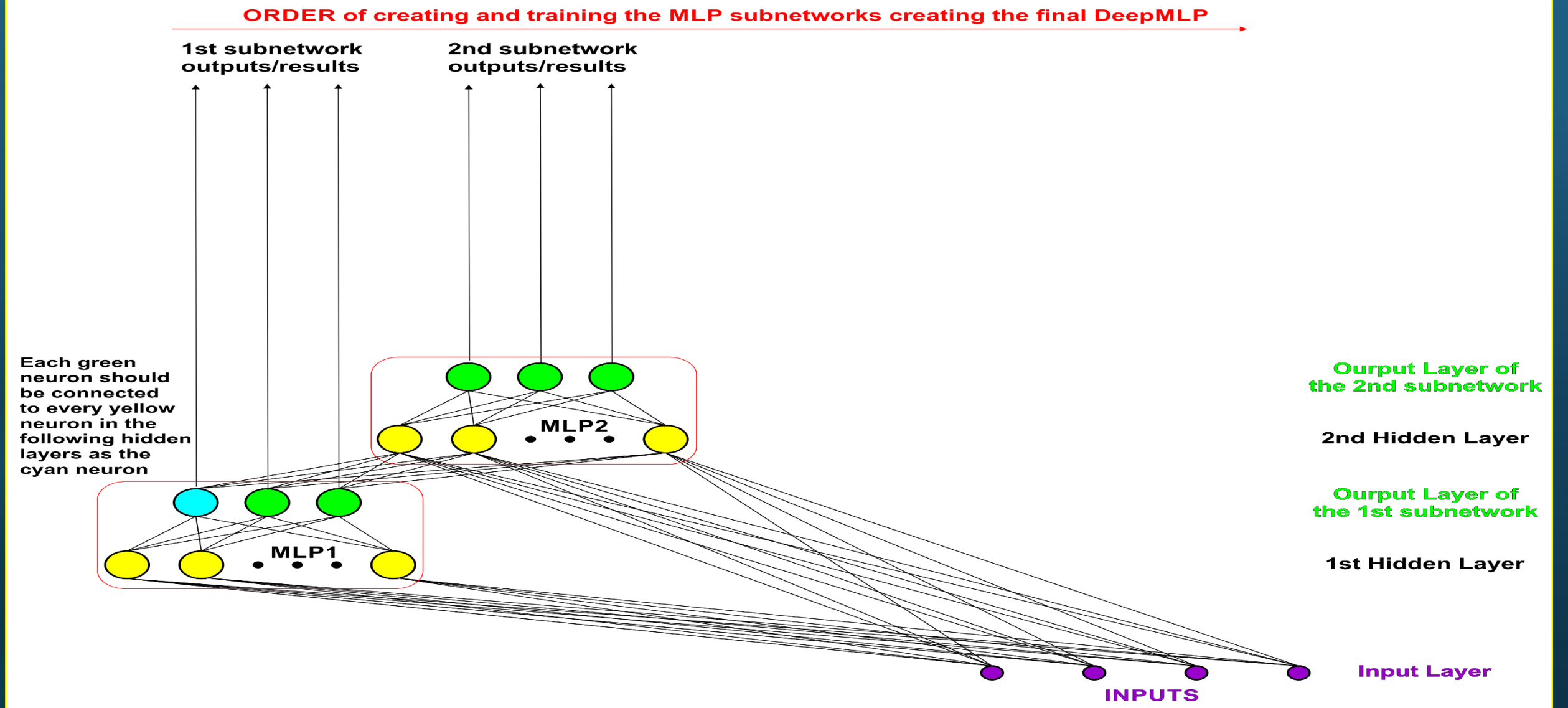
Construct a Hierarchical Deep MLP Structure



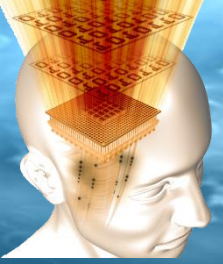


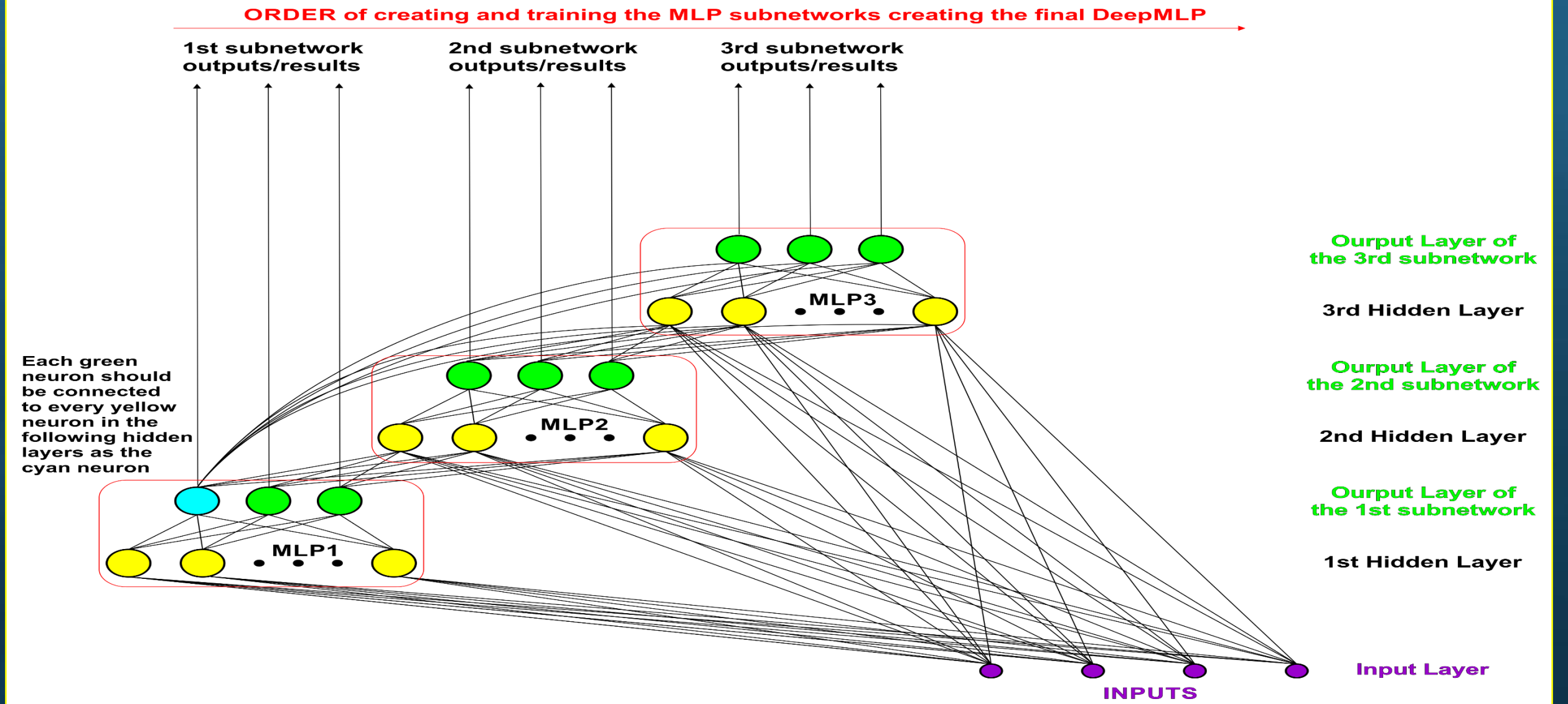
Construct a Hierarchical Deep MLP Structure



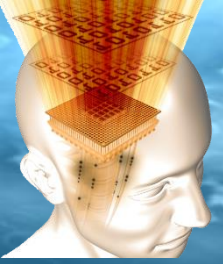


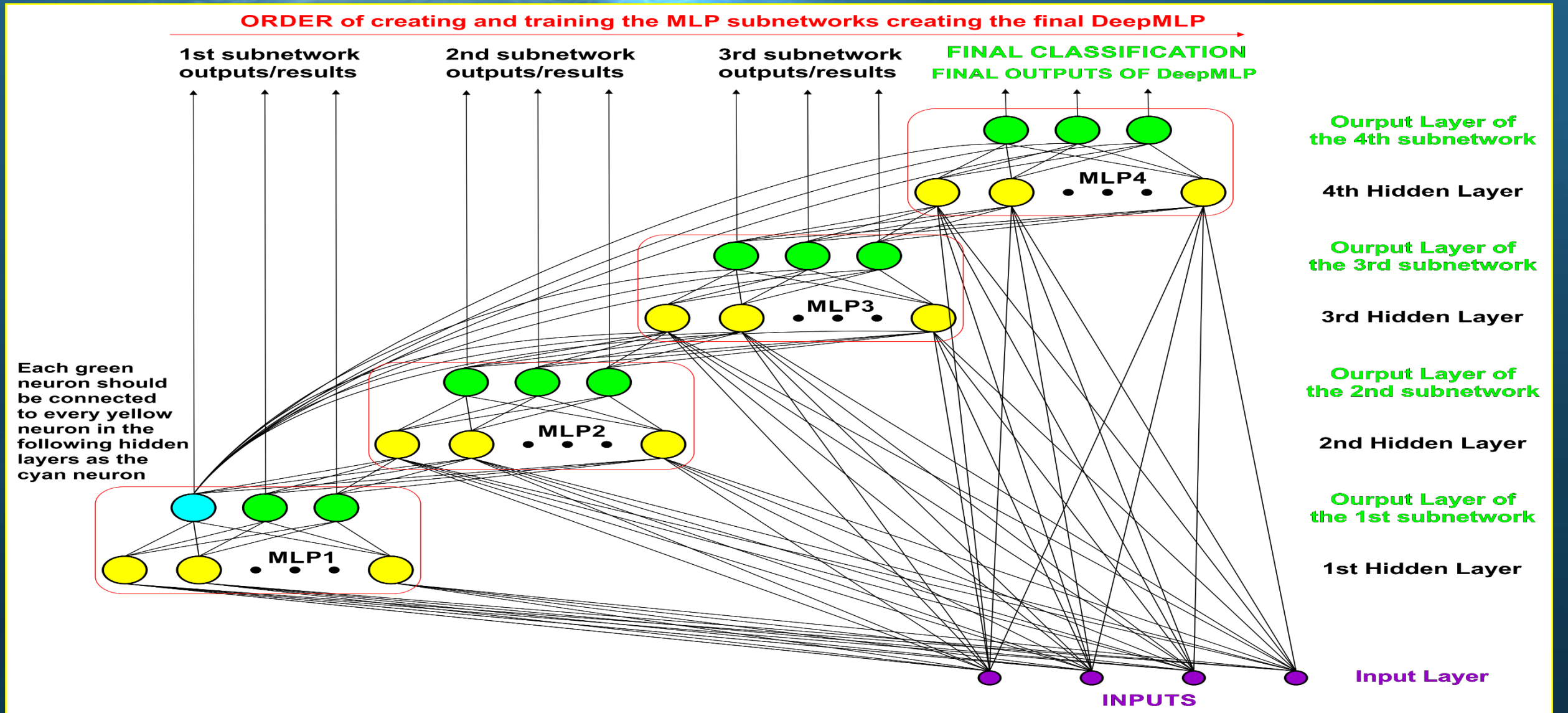
Construct a Hierarchical Deep MLP Structure

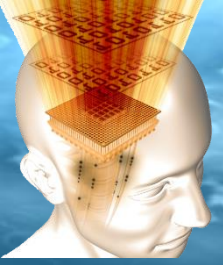




Construct a Hierarchical Deep MLP Structure





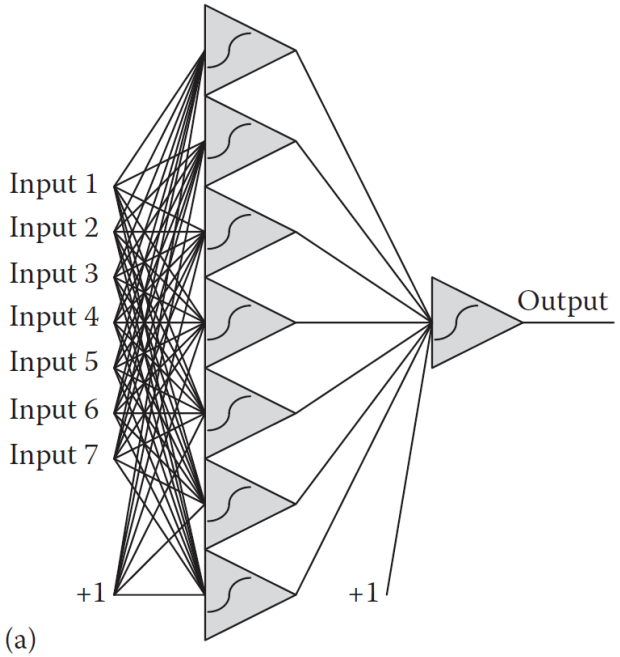


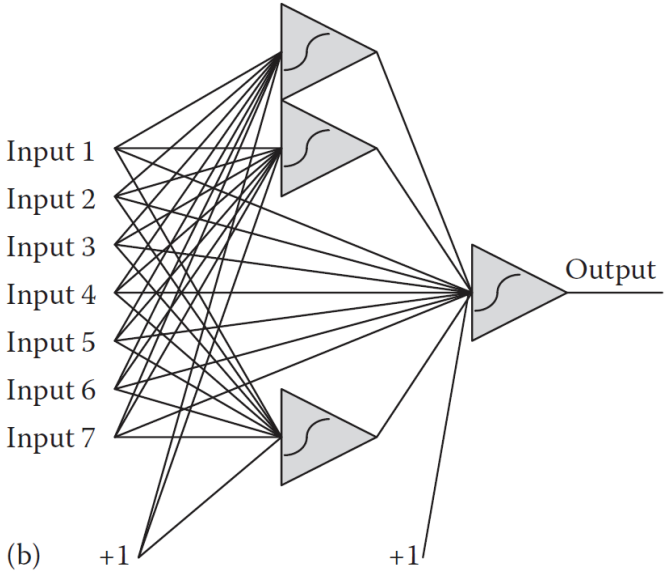
Cascade Correlated and Wide Structures

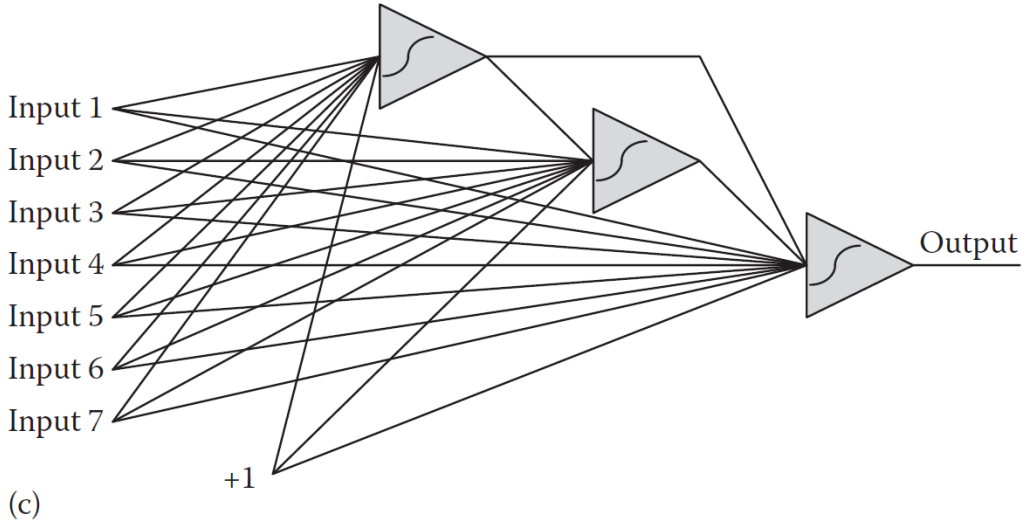
Finally, try to use different structures of connections as presented in the lectures.

Combine various approaches and/or structures and try to get the best possible classification results as possible for the tested training datasets.

Compare those approaches and try to present us the best solution!







Bibliography and References

ACADEMIC WEBSITE - ADRIAN HORZYK, PhD, DSc.

AGH University of Science and Technology in Cracow, Poland
Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering
Department of Biocybernetics and Biomedical Engineering, Field of Biocybernetics

Dossier | Research | Publications | Courses | Graduates | Consultations | Contact

AGH

LECTURES

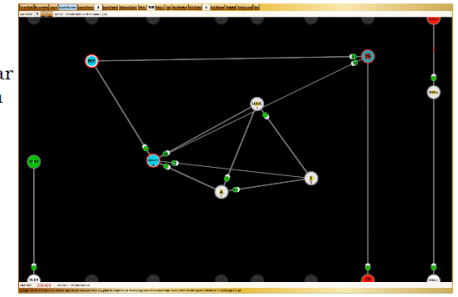
(will be renewed and expanded during the semester)

- Introduction to Artificial and Computational Intelligence
- Artificial Neural Networks, Multilayer Perceptron MLP, and Backpropagation BP
- Radial Basis Function Networks RBFN
- Unsupervised Training and Self Organizing Maps SOM
- Recurrent Neural Networks
- Introduction of Final Projects and Description of Requirements
- Associative Neural Graphs and Associative Structures
- Deep Associative Semantic Neural Graphs DASNG
- Associative Pulsing Neural Networks
- Deep Learning Strategies and Convolutional Neural Networks
- Support Vector Machines SVM
- Fuzzy Logic and Neuro-Fuzzy Systems
- Motivated and Reinforcement Learning
- Linguistic, Semantic Memories, and Cognitive Neural Systems
- Psychological Aspects of Intelligence, Human Needs, and Personality
- Writing Journal Papers

COMPUTATIONAL INTELLIGENCE

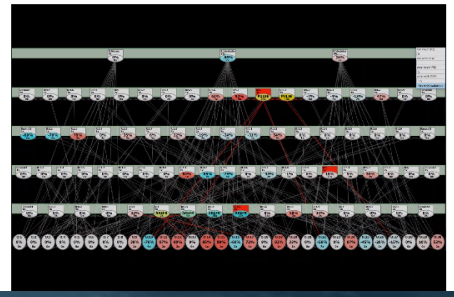
This course includes 28 lectures, 14 laboratory classes, and 14 project classes.

What is this course about?



This course is intended to give students a broad overview and deep knowledge about popular solutions and efficient neural network models as well as to learn how to construct and train intelligent learning systems in order to use them in everyday life and work. During the course we will deal with the popular and most efficient models and methods of neural networks, fuzzy systems and other learning systems that enable us to find specific highly generalizing models solving difficult tasks. We will also tackle with various CI and AI problems and work with various data and try to model their structures in such a way to optimize operations on them throughout making data available without necessity to search for them. This is a unique feature of associative structures and systems. These models and methods will allow us to form and represent knowledge in a modern and very efficient way which will enable us to mine it and automatically draw conclusions. You will be also able to understand solutions associated with various tasks of motivated learning and cognitive intelligence.

Lectures will be supplemented by laboratory and project classes during which you will train and adapt the solution learned during the lectures on various data. Your hard work and practice will enable you not only to obtain expert knowledge and skills but also to develop your own intelligent learning system implementing a few of the most popular and efficient CI methods.



Expected results of taking a part in this course:

- **Broad knowledge** of neural networks, associative and fuzzy systems as well as other intelligent learning systems.
- **Novel experience** and **broaden skills** in construction, adaptation and training of neural networks and fuzzy systems.
- **Ability to construct** intelligent learning systems of various kinds, especially deep learning solutions.
- **Good and modern practices** in modelling, construction, learning and generalization.
- **Own intelligent learning system** to use in your life or work.
- **Satisfaction** of enrollment to this course.

http://home.agh.edu.pl/~horzyk/lectures/ahdydci.php