COMPUTATIONAL INTELLIGENCE

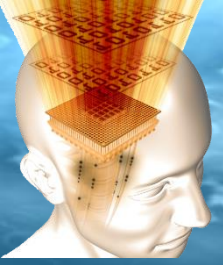
Fundamentals

AGH

Adrian Horzyk

Preface

Before we can proceed to discuss
specific complex methods we have to introduce
fundamental concepts, principles, and models
of computational intelligence
that are further used
in the complex
deep learning models.



Hebbian Learning Principle

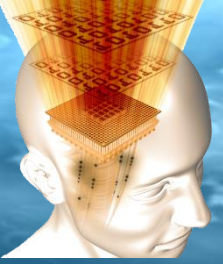
It states that „*when an axon of [neuronal] cell A is near enough to excite a [neuronal] cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, B is increased*". [D.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

This principle assumes that a connection between neuronal cells is weighted and the weight value 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.

This principle:

- is implicitly used in the most artificial neural networks today,
- is explicitly used in the LAMSTAR deep learning neural networks.

Because this principle is only half the truth about the changes in the efficiency of the synapse between cells A and B and does not describe all important synaptic processes, this issue will be discussed and extended later.



Hebb's and Oja's Learning Rule

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

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

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

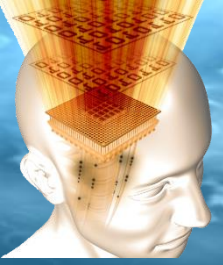
$$\Delta \boldsymbol{w} = \boldsymbol{w_{n+1}} - \boldsymbol{w_n} = \eta \cdot \boldsymbol{x_n} \cdot y_n$$

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 of a neuron to its input x is:

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

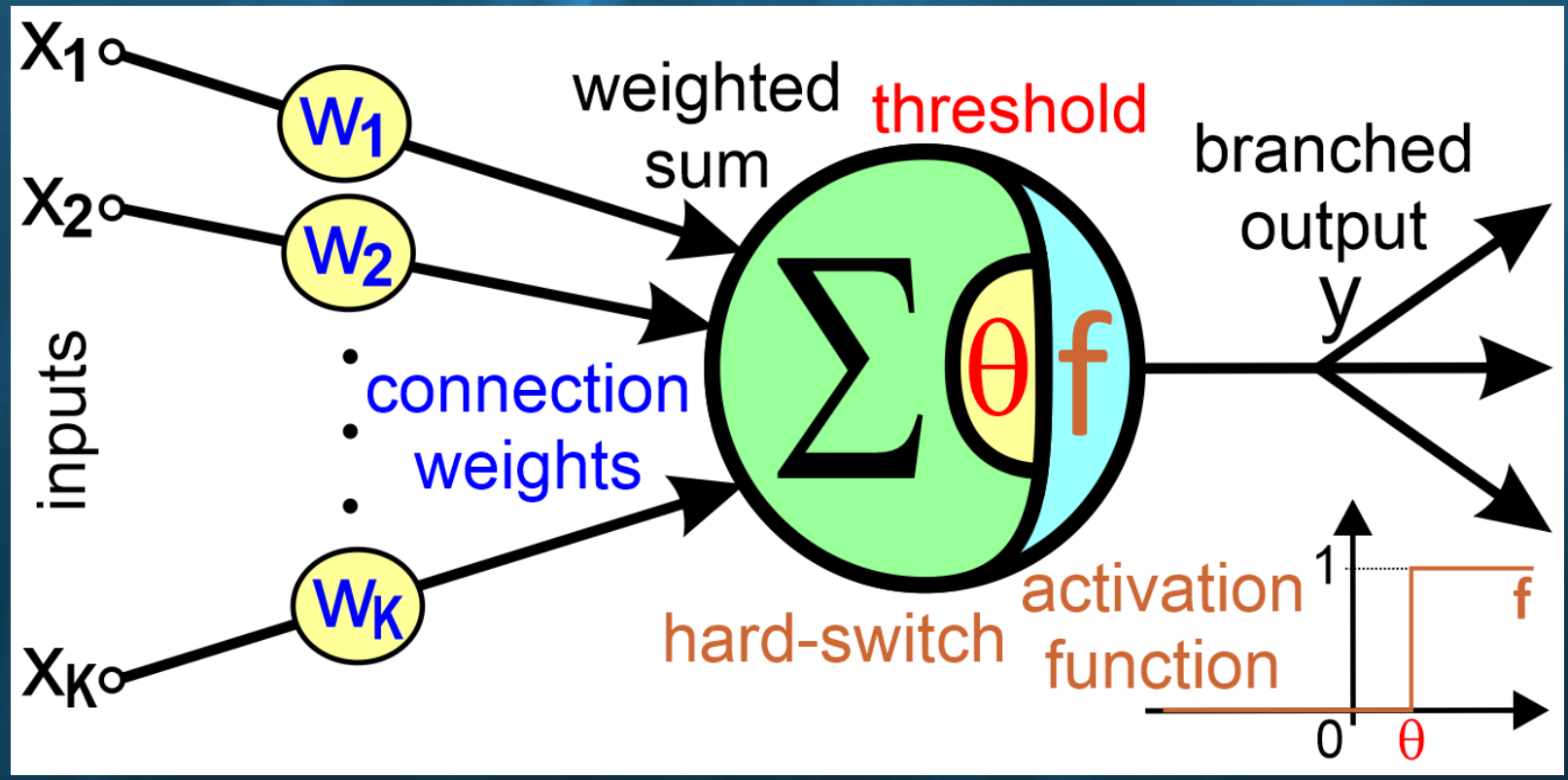
where η is a learning rate which can change over time, and n defines a discrete time iteration.



McCulloch-Pitts Model of Neuron

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.

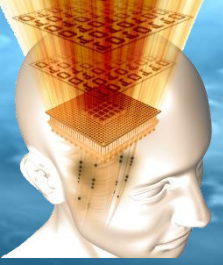
This model uses hard-switch (step) activation function f which makes the neuron active when the weighted sum S of the input stimuli X achieves the threshold θ.



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

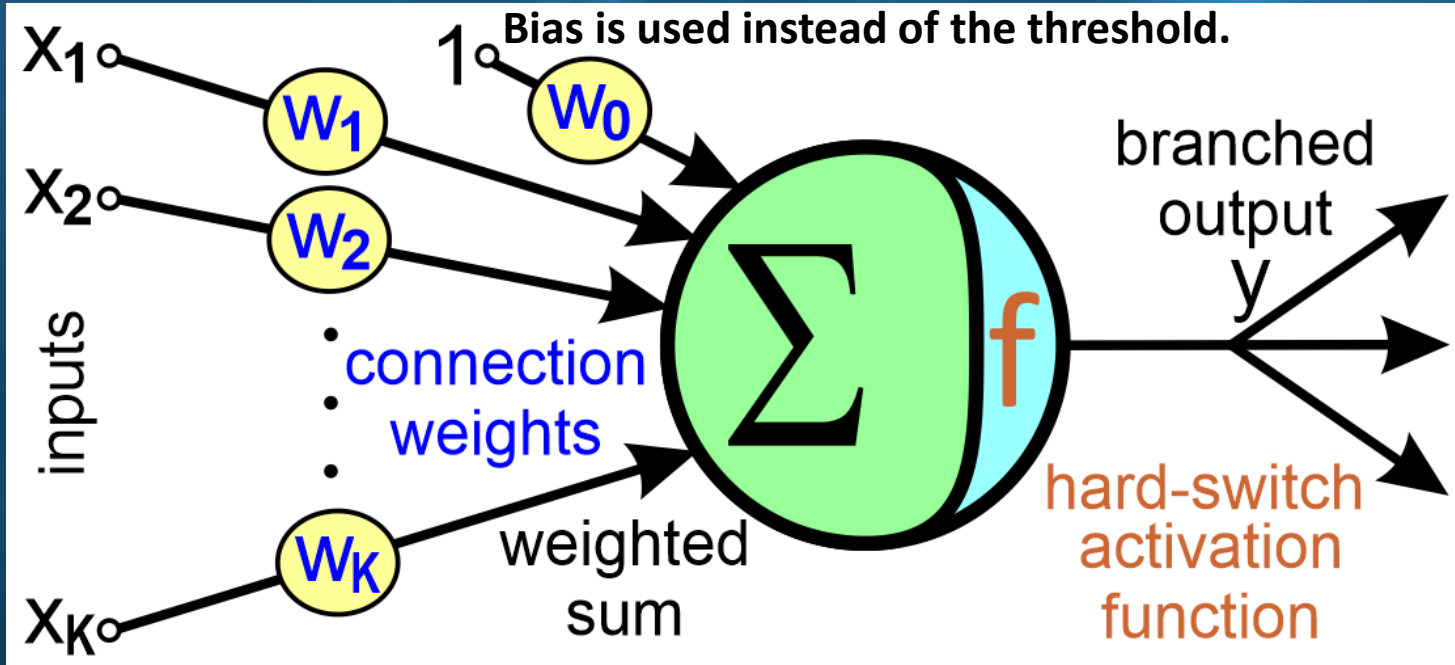
$$S = \sum_{k=1}^{K} x_k \cdot w_k$$

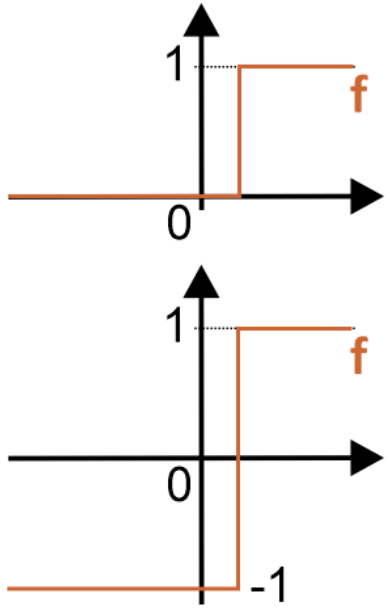
$$X = [x_1, \ldots, x_k]$$

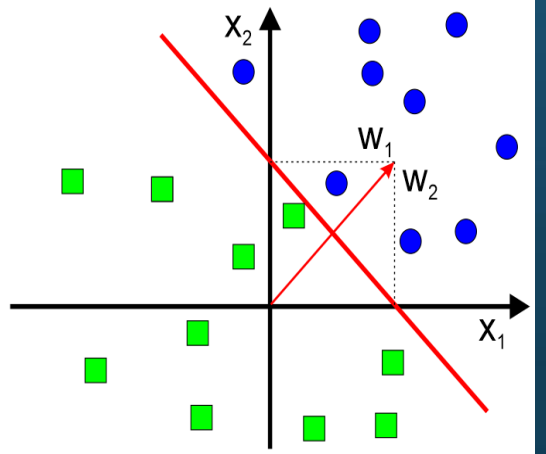


Hard-Switch Perceptron

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 function f:

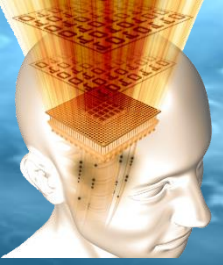






The decision boarder determined by the perceptron

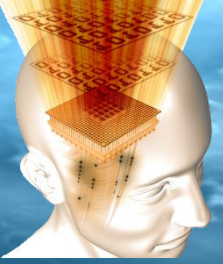
$$y = f(S) = \begin{cases} 0 & S < 0 \\ 1 & S \geq 0 \end{cases} \qquad S = \sum_{k=0}^{K} \mathbf{x}_k \cdot \mathbf{w}_k \qquad X = [\mathbf{x}_1, \ldots, \mathbf{x}_k] \qquad \mathbf{x}_0 = 1$$



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, …, 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 and 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,…,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.



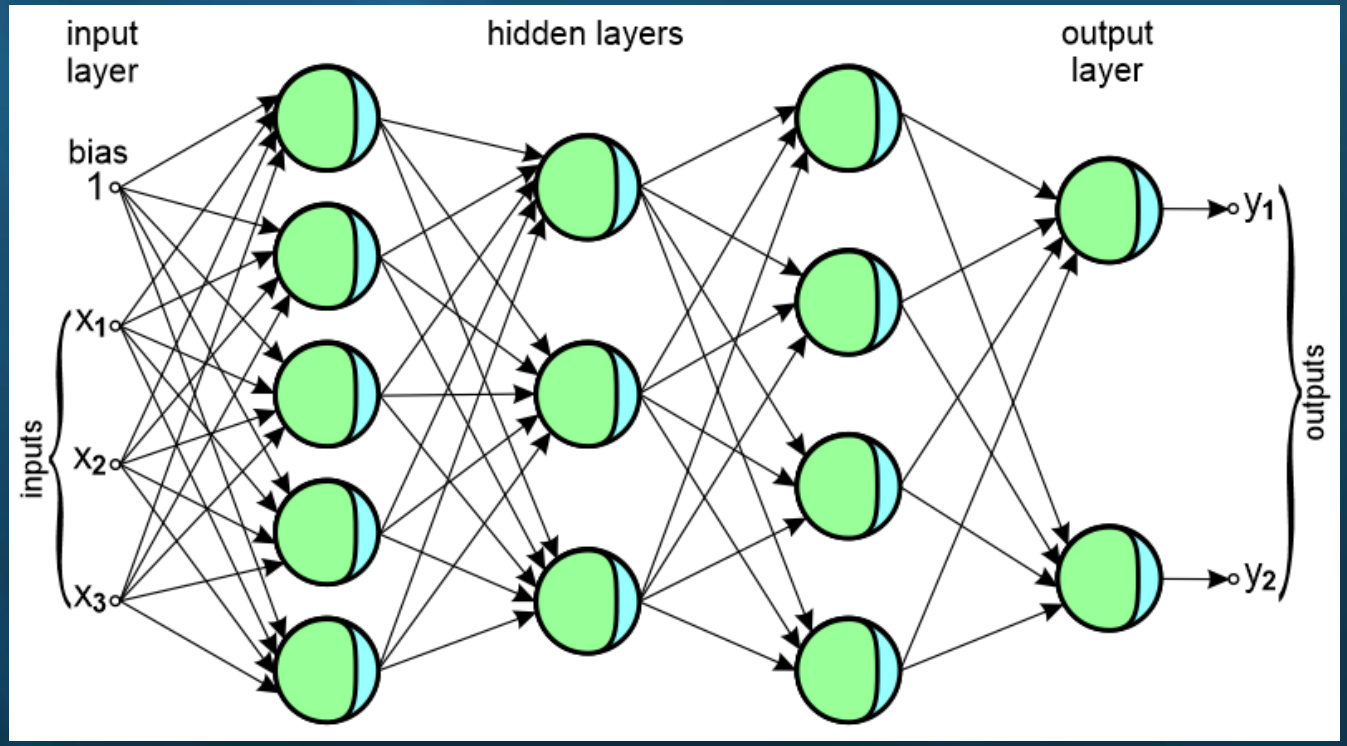
Single and Multi-Layer Perceptron

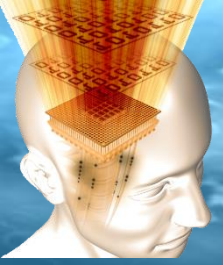
A group of perceptrons organized in a single layer can be used for the multi-classification which means the classification of input vectors into a few classes simultaneously.

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.

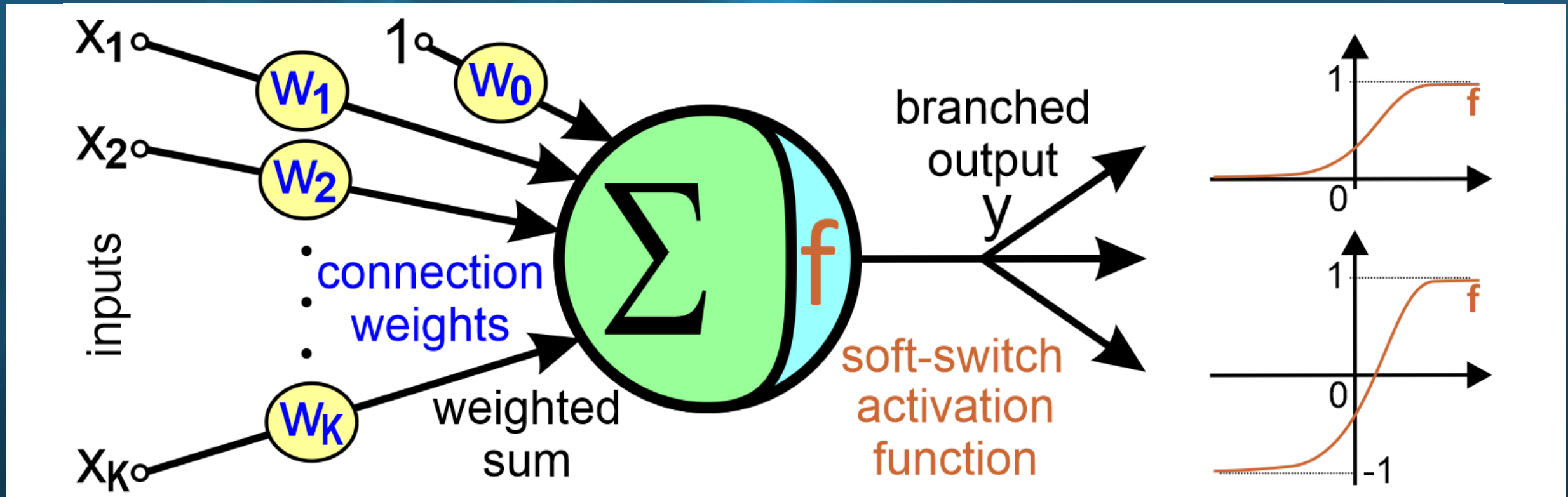
The first layer is called input layer, the last one is called output layer, and all the layers between them are hidden as shown in the figure:



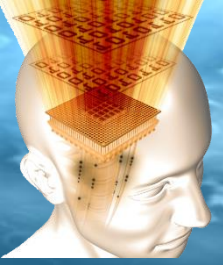


Soft-Switch Perceptron

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 Neuron Adaptation

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 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:

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

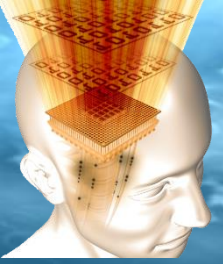
The correction of the weight for differentiable activation function f is computed after:

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

where f' is the differential of the function f.

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

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



MLP, BP, and CNN

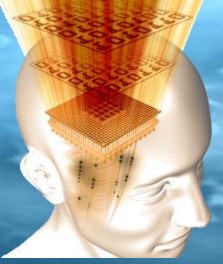
The continuous, soft-switching nature of the sigmoid function allows it to be differentiable everywhere. This is necessary for several learning algorithms, such as Backpropagation or Convolutional Learning.

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

Multi-layer perceptron cannot use linear soft-switch activation function because each multi-layer linear perceptron network 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.

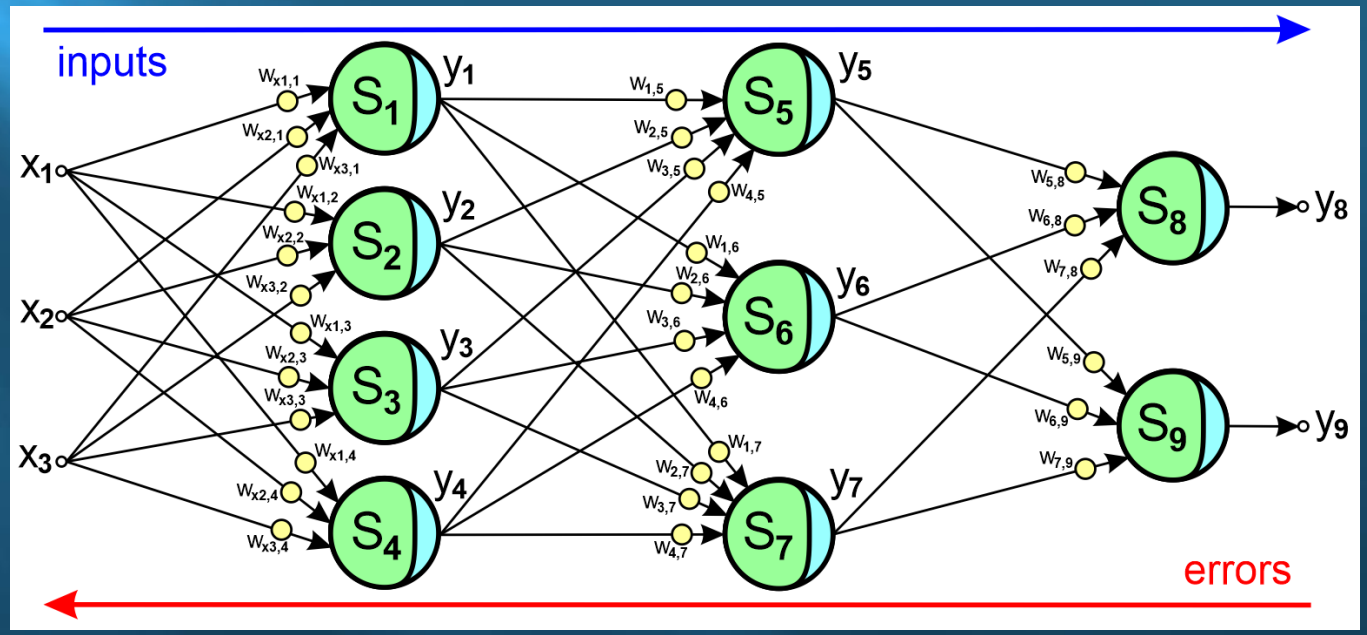
The BP algorithm is too slow to satisfy the machine learning needs, but it was rehabilitated later on (in 1989) when it became the learning engine of the far faster and the most popular Convolutional Deep Learning Neural Networks (CNN).



Backpropagation Algorithm

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

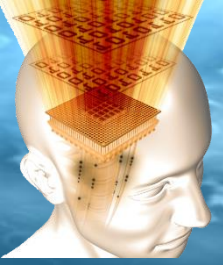
1. The input propagation phase propagates the inputs throughout all hidden layers to the output layer neurons. In this phase, neurons make summation of weighted inputs taken from the neurons in the previous layer.
2. The error propagation phase propagates back the errors (delta values) computed on the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

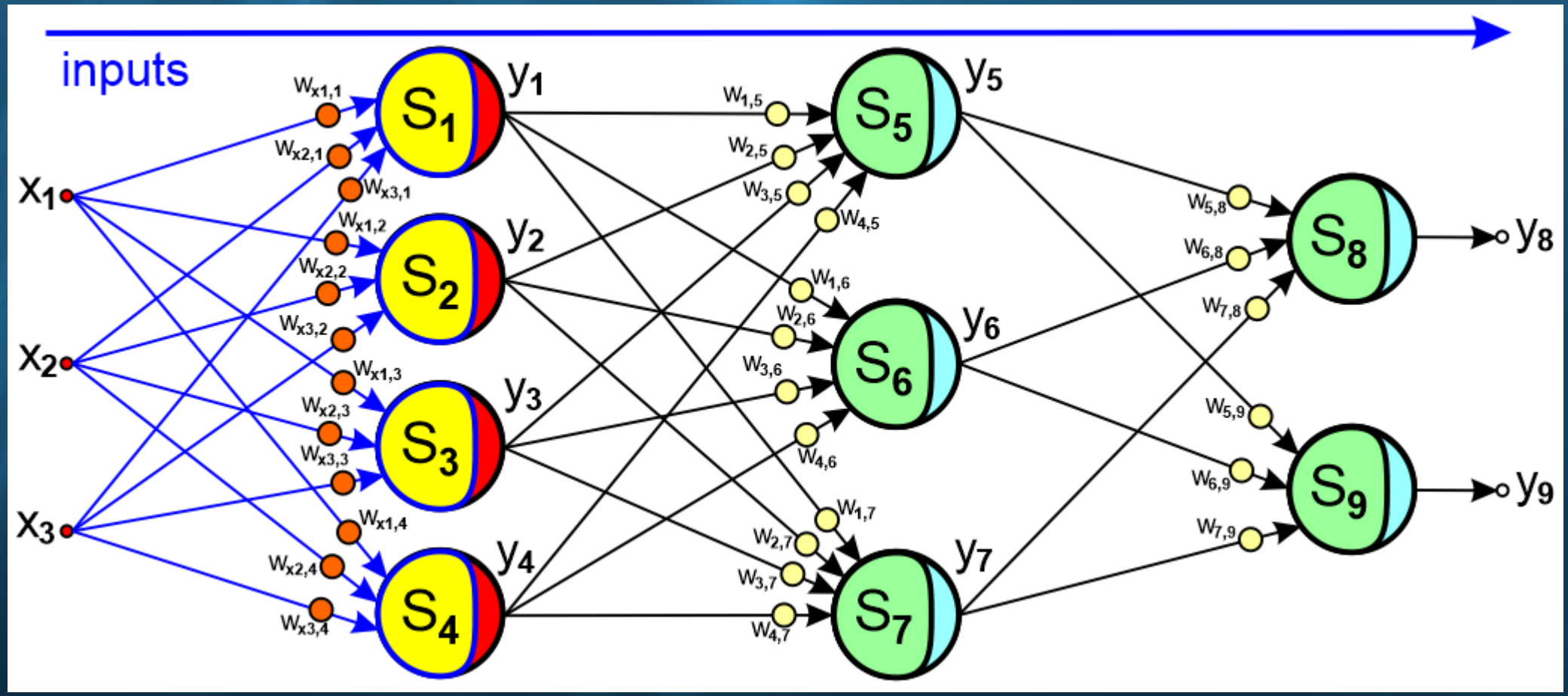
This algorithm is executed until the mean square error computed for all training samples is less than the desired value or to a given maximum number of cycles.

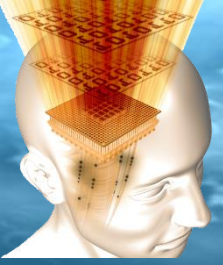


Backpropagation Algorithm

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:

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

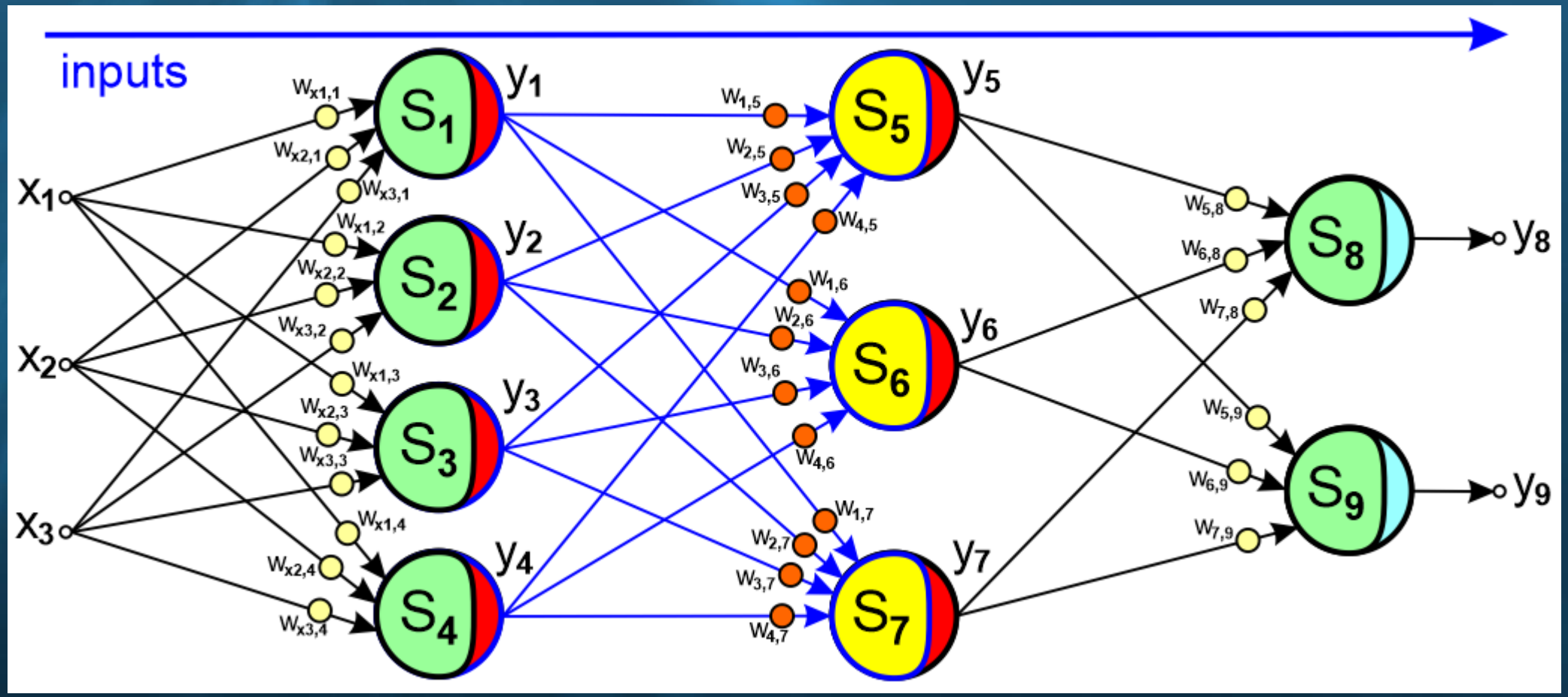


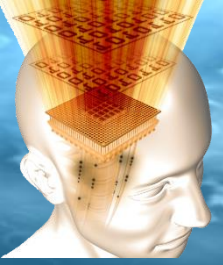


Backpropagation Algorithm

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:

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

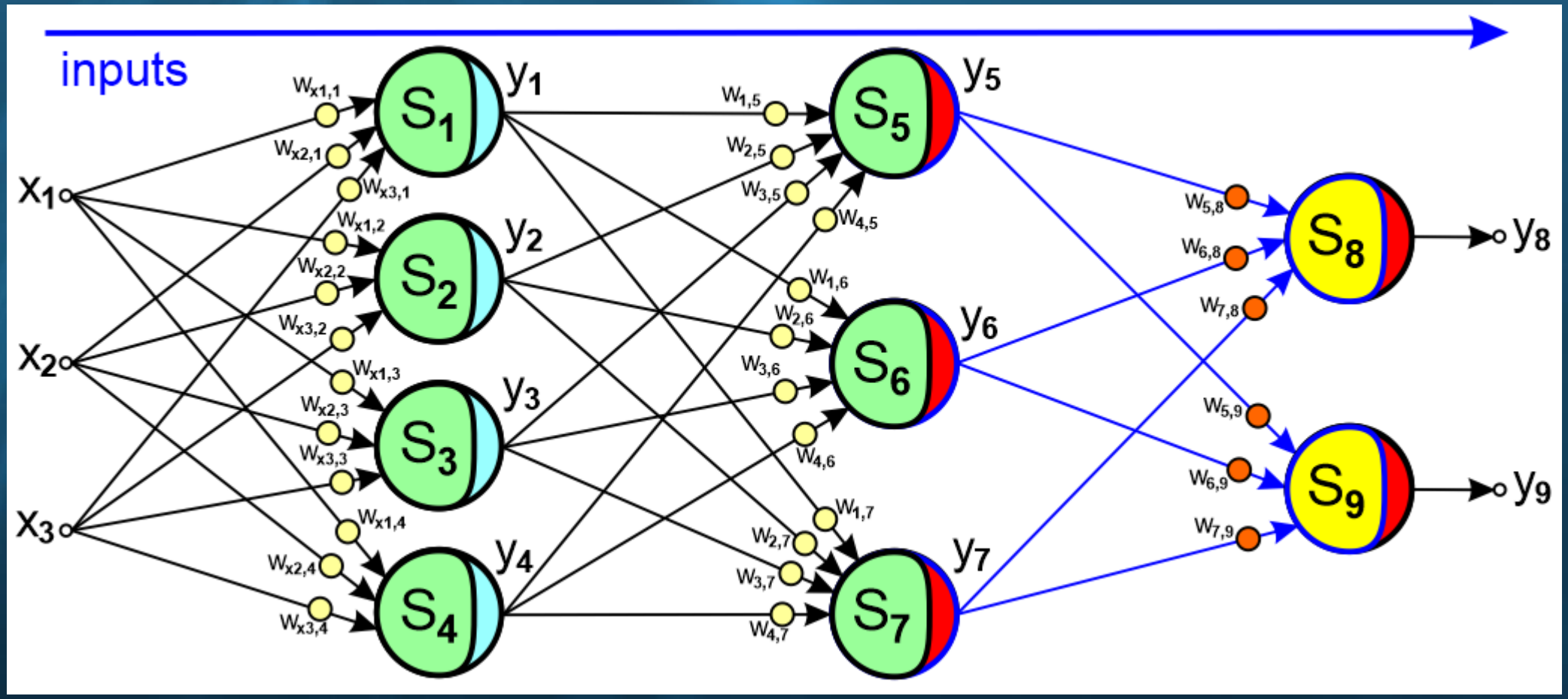


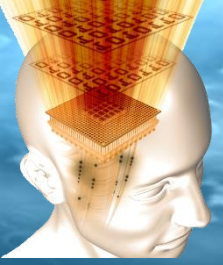


Backpropagation Algorithm

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:

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

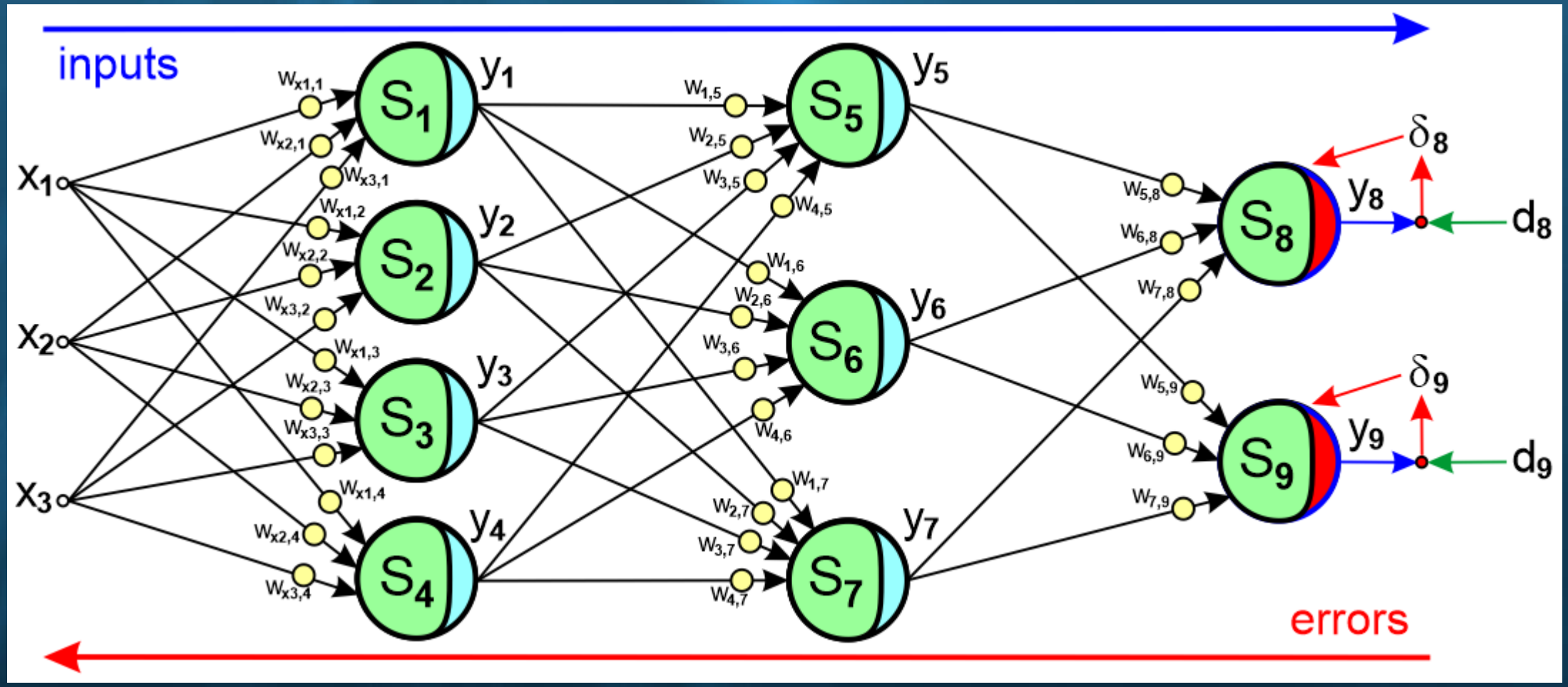


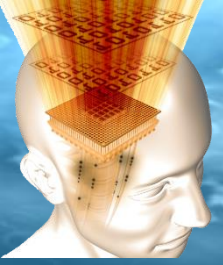


Backpropagation Algorithm

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.

$$\delta_n = d_n - y_n$$



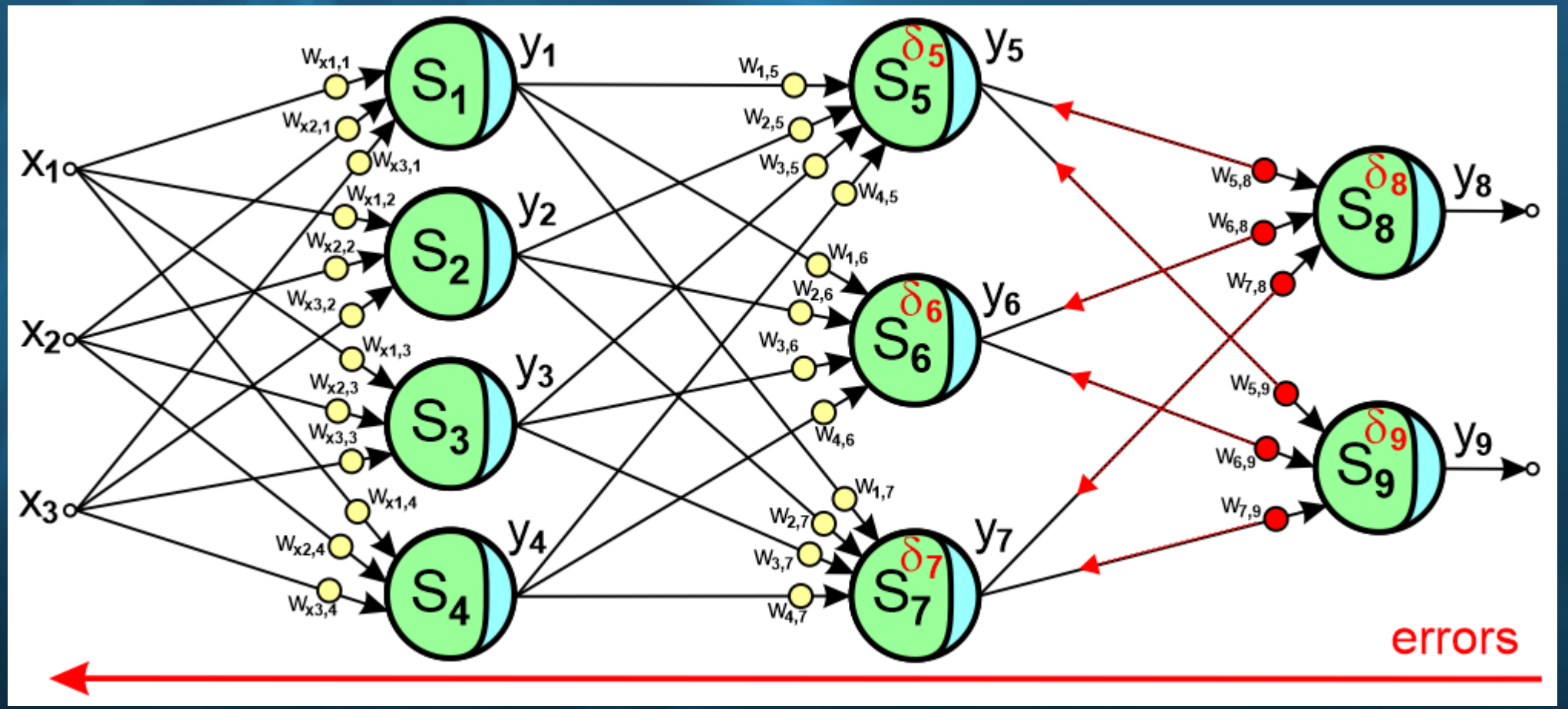


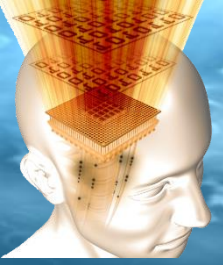
Backpropagation Algorithm

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:

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

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



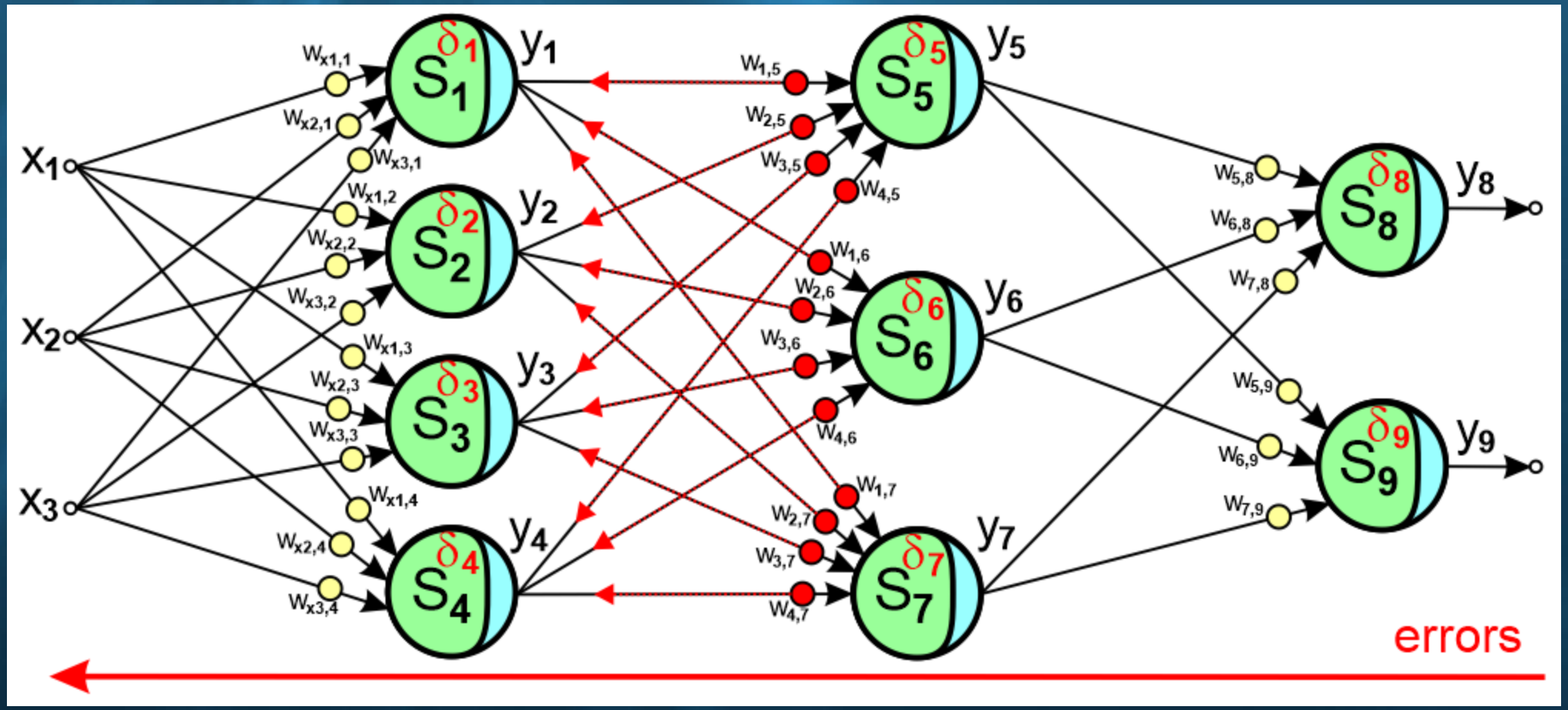


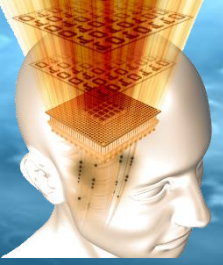
Backpropagation Algorithm

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:

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

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

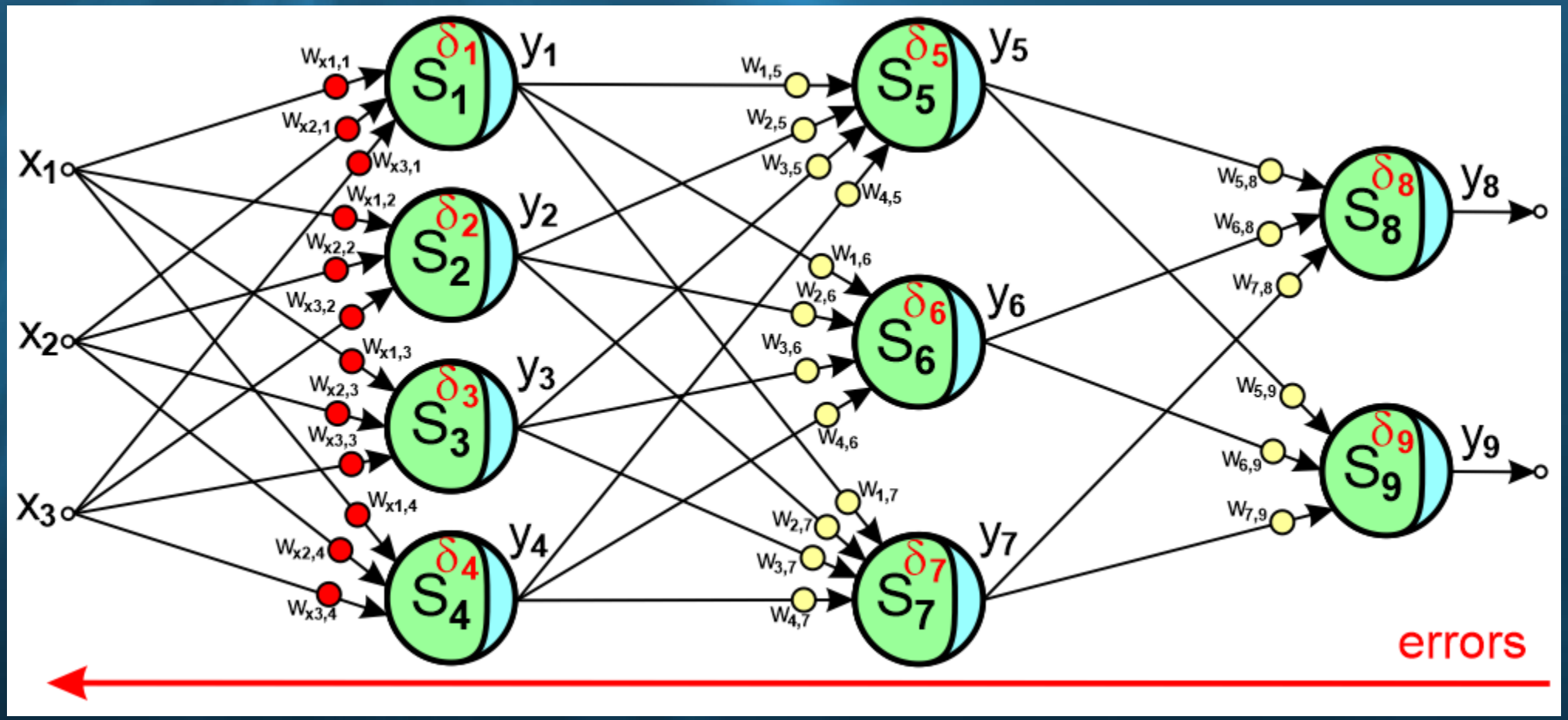


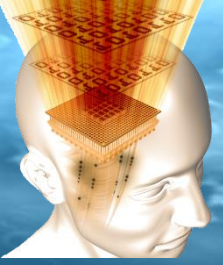


Backpropagation Algorithm

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

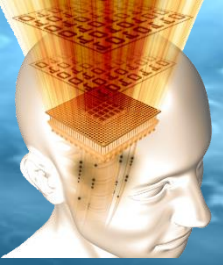
The number of hidden layer neurons should be higher rather than lower. However, for simple problems, one or two hidden layers may suffice.

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 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 -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.



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:

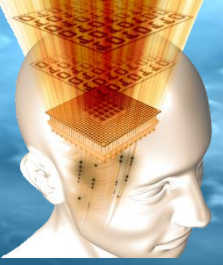
$$\Delta \mathbf{w}_{k,n}^{p} = \alpha \cdot \Delta \mathbf{w}_{k,n}^{p-1} + \eta \cdot \delta_n \cdot f'\left(\sum_{k=0}^{K} \mathbf{x}_k \cdot \mathbf{w}_k\right) \cdot \mathbf{x}_k = \alpha \cdot \Delta \mathbf{w}_{k,n}^{p-1} + \eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot \mathbf{x}_k$$

- 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.



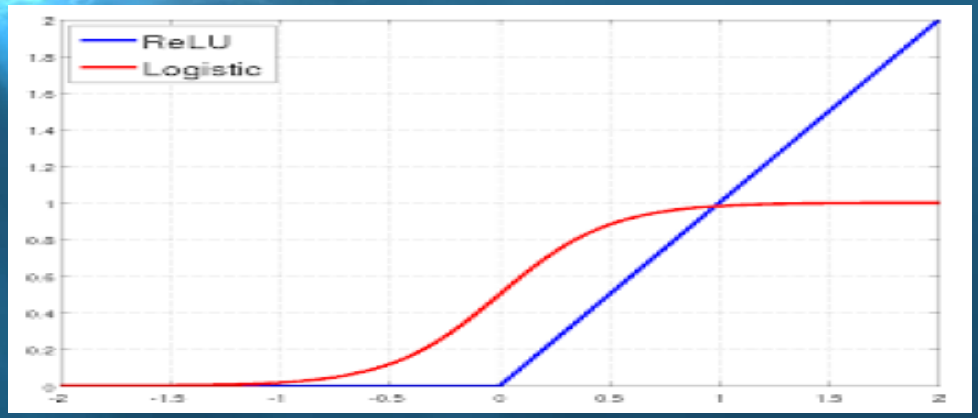
Overcome Convergence Problems

To overcome convergence problems of the backpropagation algorithm we can:

- Change the range of the sigmoid function from [0, 1] to [-1, 1].
- Modifying the 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.
- Reduce the number of inputs to overcome the curse of dimensionality problem.
- Use cross-validation to avoid the problem of over-fitting.

Rectified Linear Units (ReLU)

We can also use Rectified Linear Units (ReLU) to eliminate the problem of vanishing gradients.

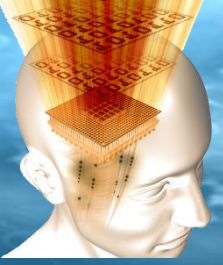


ReLU units are defined as: f(x) = max(0, x) instead of using the logistic function.

The strategy using ReLU units is based on training of robust features thanks to sparse (less frequent) activations of these units.

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

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

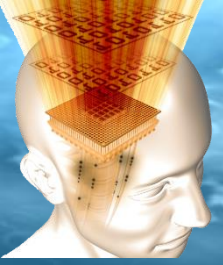


K-fold Cross-Validation

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



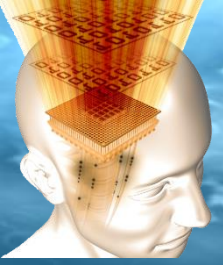
K-fold Cross-Validation

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 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

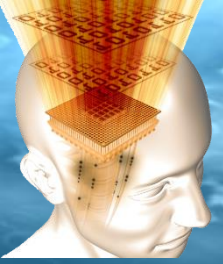
10-FOLD	SUBSETS OF TRAINING PATTERNS									
STEPS	1	2	3	4	5	6	7	8	9	10
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN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

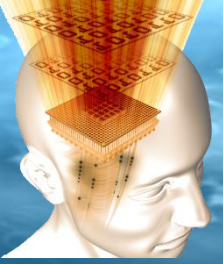
The way of selection of the test patterns in each training step should be representative and proportional from each class point of view regardless of the cardinality of classes!
We have to consider how the training data are organized in the training dataset:

- 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:

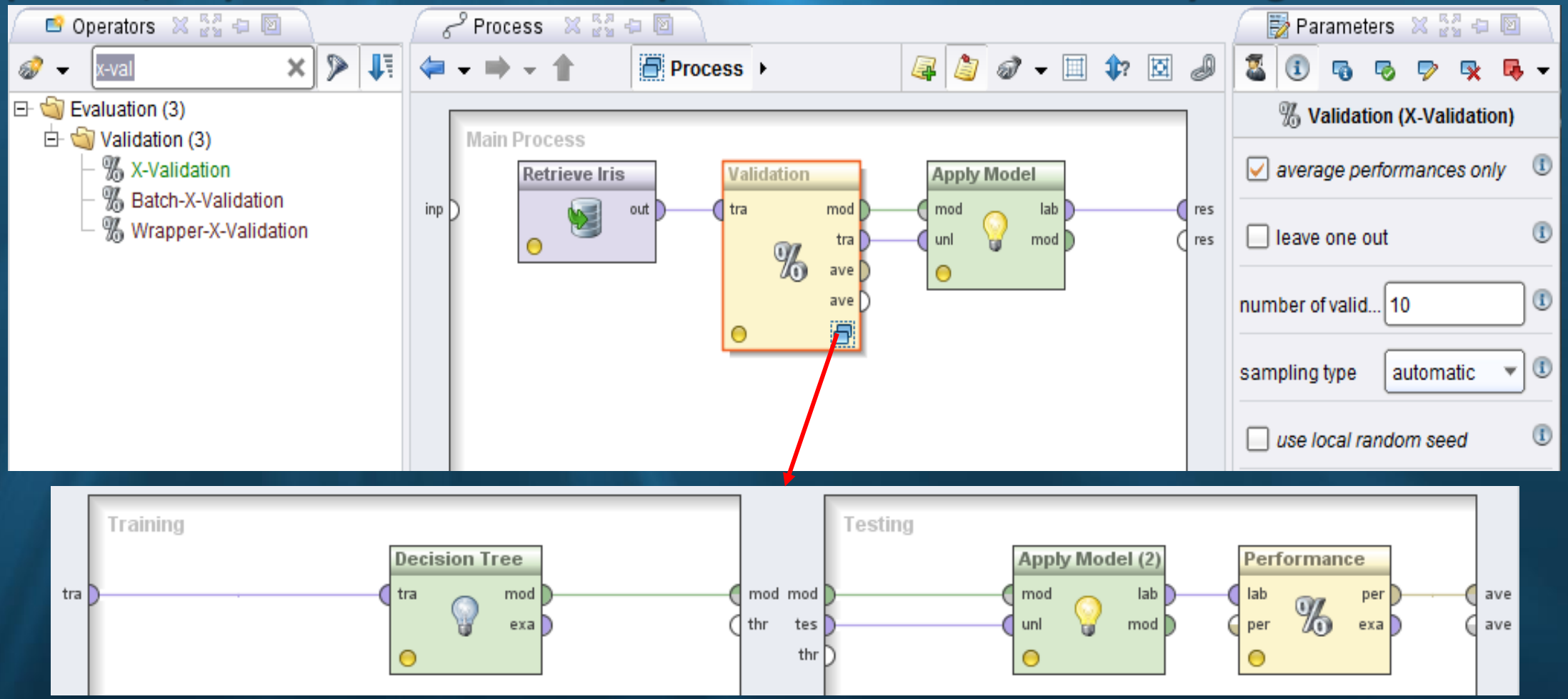
5-FOLD	SUBSETS OF TRAINING PATTERNS																																					
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN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

Rapid Miner K-fold Cross-Validation

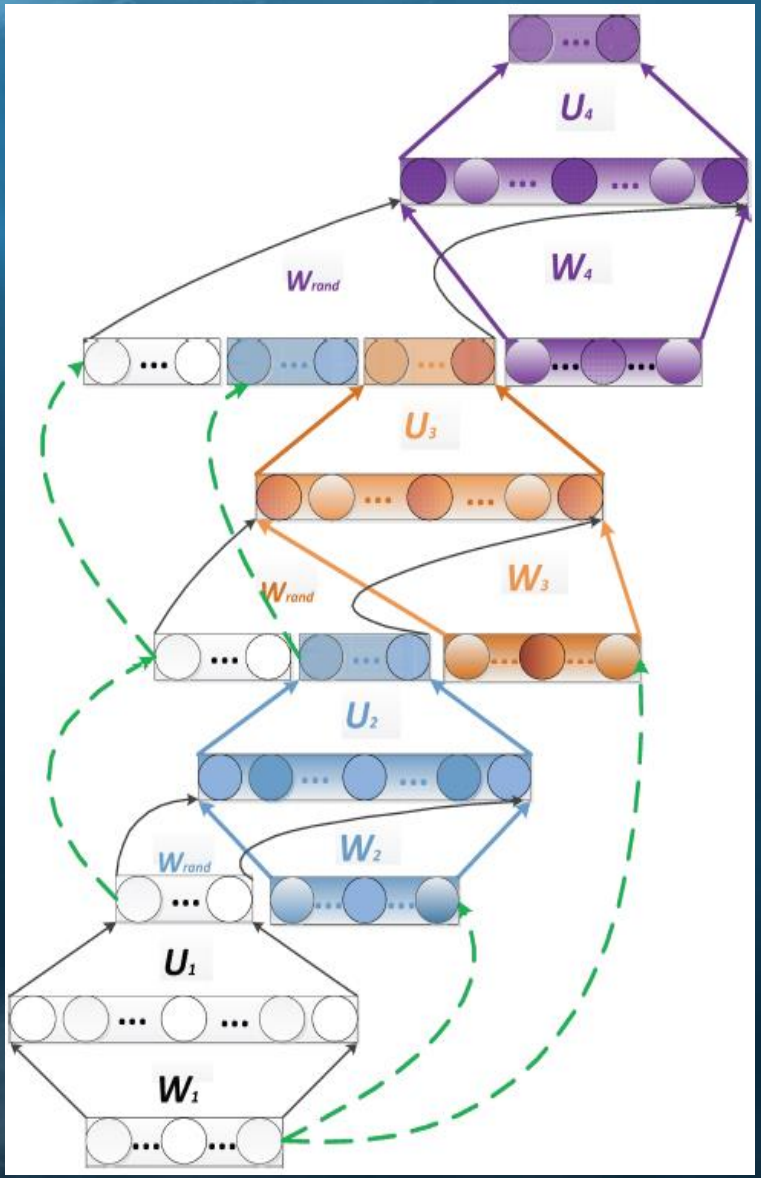
This computational tool also supplies us with the ability to use Cross-Validation during MLP adaptation, so you can check and compare achieved results and try to get the better ones:



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 tiny changes of weights in the neuron layers that are far away from the output of the MLP network.

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.

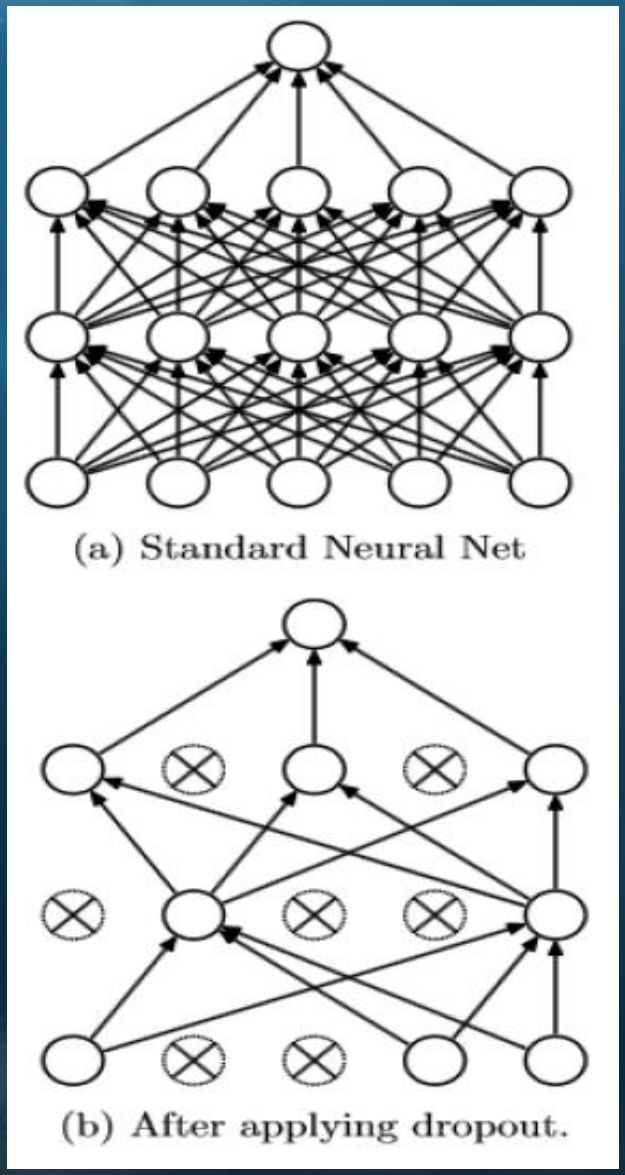


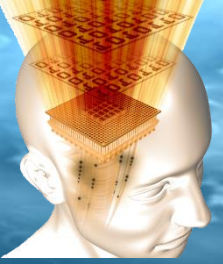
Dropout Regularization Technique

We can also use regularization techniques called dropout.

This training strategy selects only these neurons which are already the best adapted in various layers and performs training only to these neurons and their weights. This technique prevents neural networks from overfitting and also speeds up training. It also prevents other neurons from spoiling their weights parameters which can be useful for other training patterns.

Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. JMLR 2014

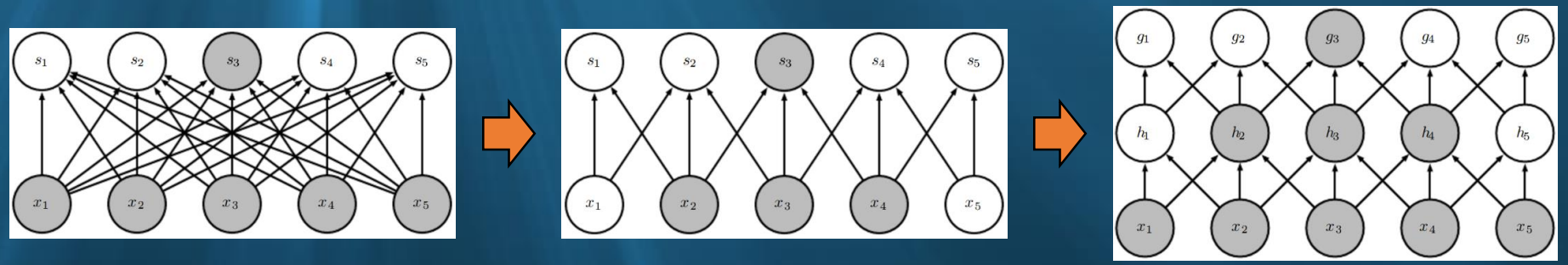




Deep Learning Strategies

Deep learning strategies assume the ability to:

- 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.

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

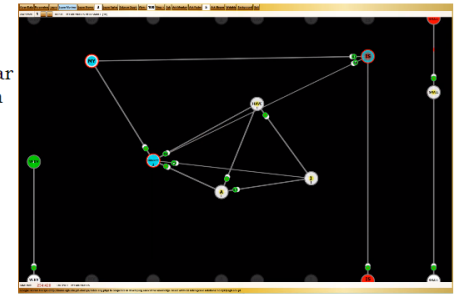
(will be renewed and expanded during the semester)

- 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

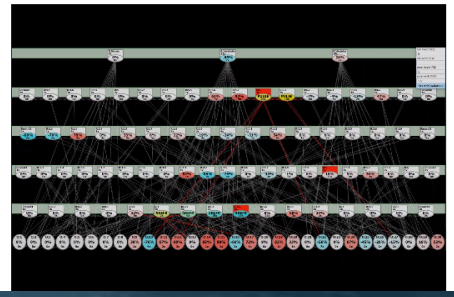
This course includes 28 lectures, 14 laboratory classes, and 14 project classes.

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