How to develop an intelligent ANN?

- Once properly trained, an ANN can become intelligent.
- The steps to developing a well-trained ANN are:
  - Select an appropriate ANN architecture
  - Select a the best training algorithm to be employed for the task
  - Select suitable activation functions
  - Train the ANN until it becomes intelligent This is also known as ANN learning process.

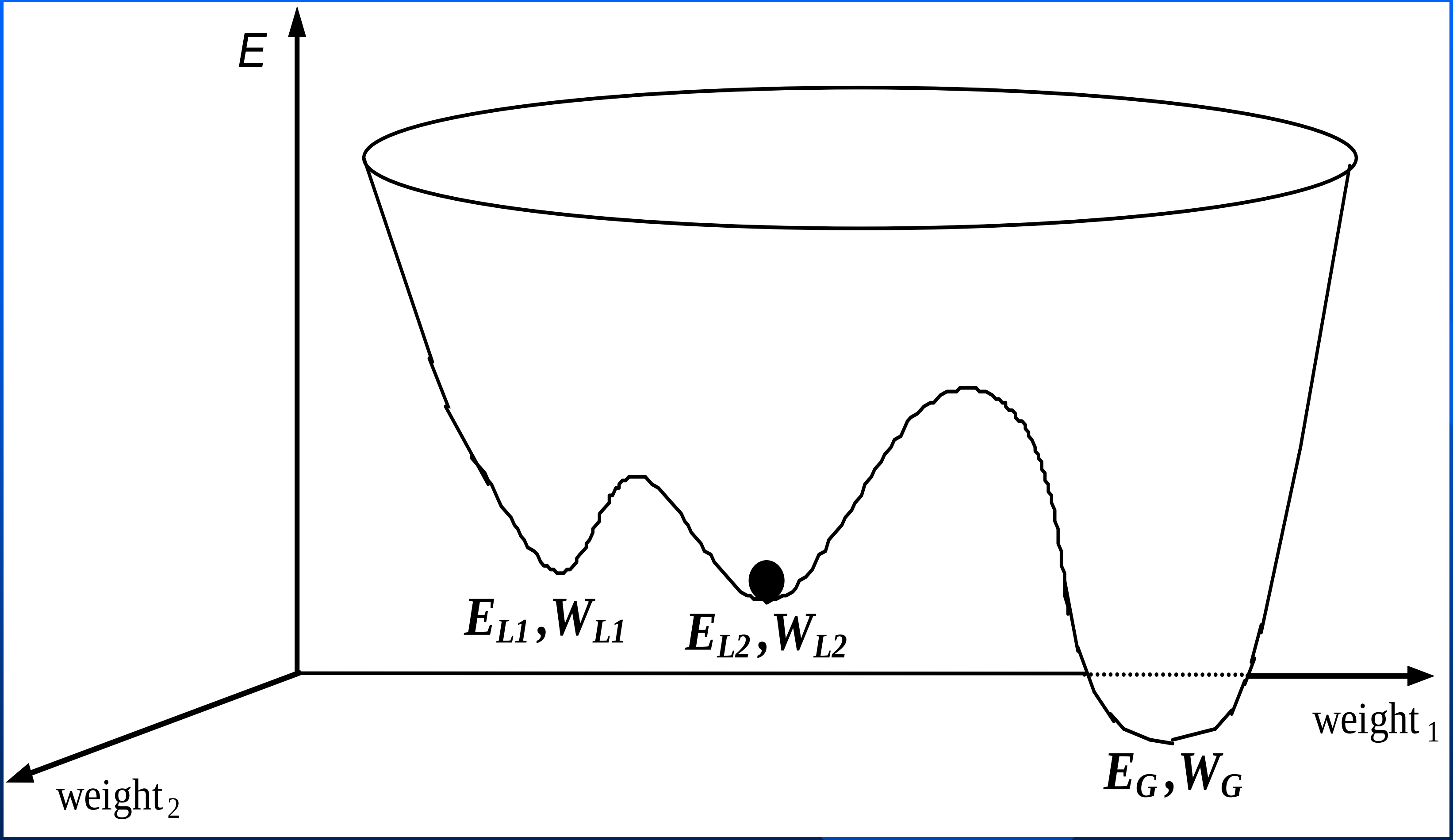
What is the objective of ANN learning?

- Theoretically, training or learning in ANN serves TWO objectives:
  - ANN learning is a process of tuning or updating ANN weights using appropriate learning algorithm until a satisfactory discrepancy (on the outputs) is achieved. At this point, optimum ANN weights are obtained and global minimum *i.e. generalization* is reached. During learning, an ANN must be able to avoid the traps of local minima in its quest towards the global minimum.
  - To obtain optimum ANN structure. This is accomplished by optimum number of neurons in each layer of the ANN, i.e. the input and hidden layer neurons.

ANN Generalization Capability: Local Minima Vs Global Minimum

- ANN learning process intend to attain a global minimum, which is a location (on an error surface) corresponding to the least error value. Such an ANN would have a generalization capability.
- Analogy of minima
  - ANN learning can be thought of as a process of travelling to reach a destination. Consider a traveler, travelling from A to B where B is the ultimate destination, i.e. the global minimum.
  - However, during his travel, he may get stalled due of few problems such as car breaks down, illness, etc. The problem are traps which hinders him from reaching the intended destination on time. These traps are the local minima.
- An ANN which is stuck in any one of the local minima will not be able to perform give correct output for presented inputs.

Local Minima Traps



ANN gets stuck in a local minimum trap at location (E_{l2},W_{l2}) and hence has not reached the global minimum (E_G,W_G)

How to ensure global minimum?

Repeat training using different initial weights, i.e. start training at different locations on the error surface.

Test all the ANNs with the same set of test data.

ANN with the least error is assumed to have reached the global minimum. It is regarded as the best-performed ANN.

Store the ANN weights for further use to solve the particular task.

When to stop an ANN training?

- ANN training stops based on one of 3 stopping conditions:
  1. Number of epochs. When max epochs reached.
  2. Error goal. When training error is equal or less than the error goal.
  3. Validation or cross-validation error performance. When there is no further improvements in validation error after a few consecutive epochs.

Which stopping condition is the best?

- Validation condition is the best.
- Why?
  - The condition automatically resumes ANN learning until a mature ANN is produced. It also stops ANN learning before the system is overtrained. Hence, it avoids the production of <u>immature</u> as well as <u>overtrained</u> ANN.

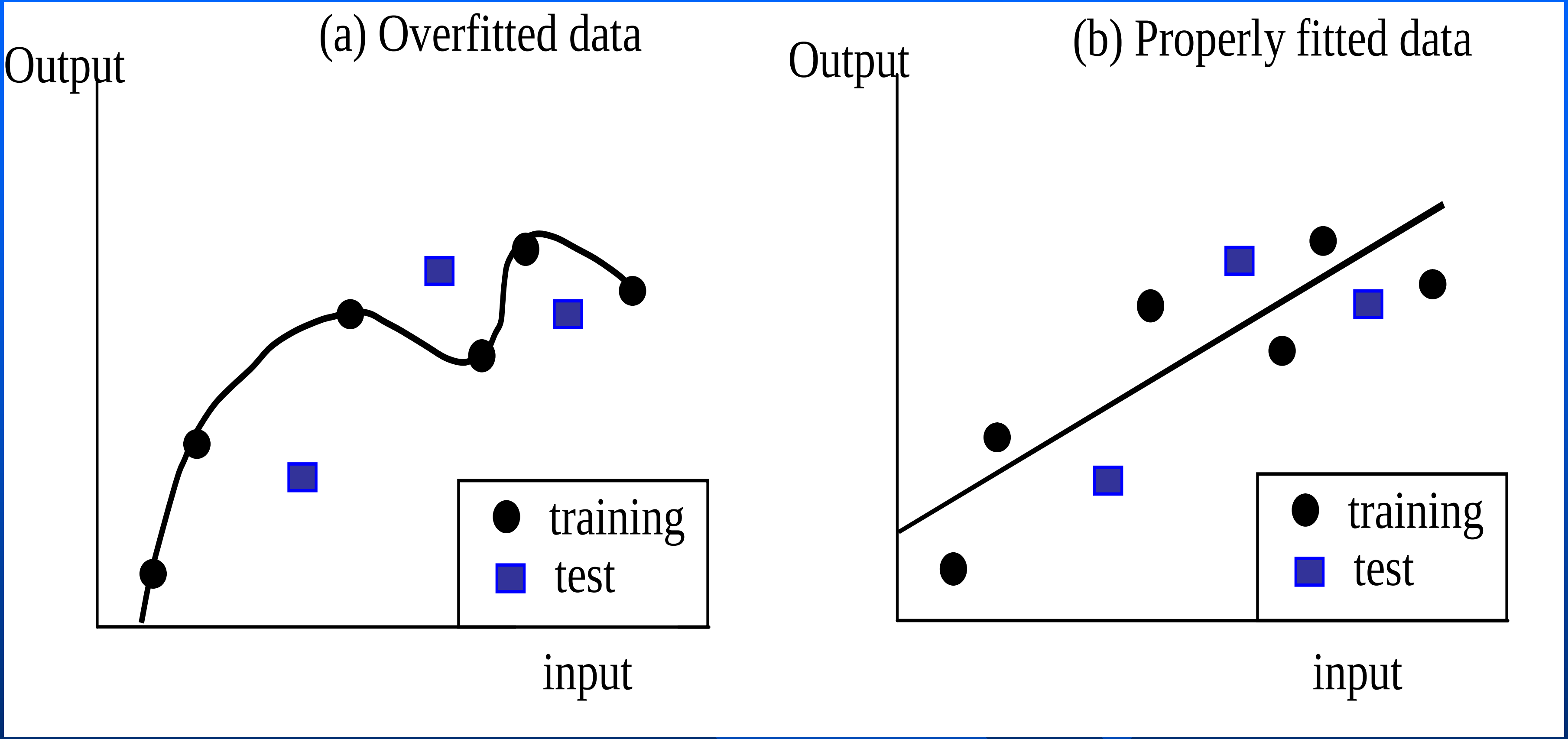
Immature & Overtrained ANN

- What is an immature ANN?
  - It is an ANN which has not learnt in sufficient period of time (i.e. epochs). It will not be able to provide accurate output, i.e. not capable of generalizing.
- A mature and well-trained ANN ensures the production of an ANN that is able to <u>generalize</u>.
- What is an overtrained ANN?
  - It is an ANN which has learnt too much (too many training epochs). It can no longer concentrate at learning, i.e. it has reached a saturation in learning and hence it has <u>no generalization</u> capability.
- An overtrained ANN leads to the problem of <u>overfitting</u>.

Problem: Overfitting of data

- Data overfitting is a problem whereby an ANN tightly fit the input samples.
- This causes the ANN to be inflexible, and hence not able to generalize when presented with new data.
- This could be caused by few factors such as:
  1. too many input samples presented during a training or learning process.
  2. too many input features or number of inputs presented in each input dataset.
- Problem 1. can be eliminated by trial-and-error on the number of optimum input samples.
- Problem 2. can be eliminated by using statistical method for correlated data elimination.

Problem: Overfitting of data



(a) ANN overfits the data

(b) ANN properly fits the data, establishing a best-fit

Non-Linear Neural Network Architecture

- There are many types of non-linear NN.
- The simplest non-linear NNs are multilayer NN, one which has more than just the input and output layers of neurons. They could be a feedforward NN, which means information is passed in a forward manner.
- Other non-linear NNs are single-layer NNs with backpropagation or recurrent connections. Besides feed forward links, they also have feedback links for passing information backward.

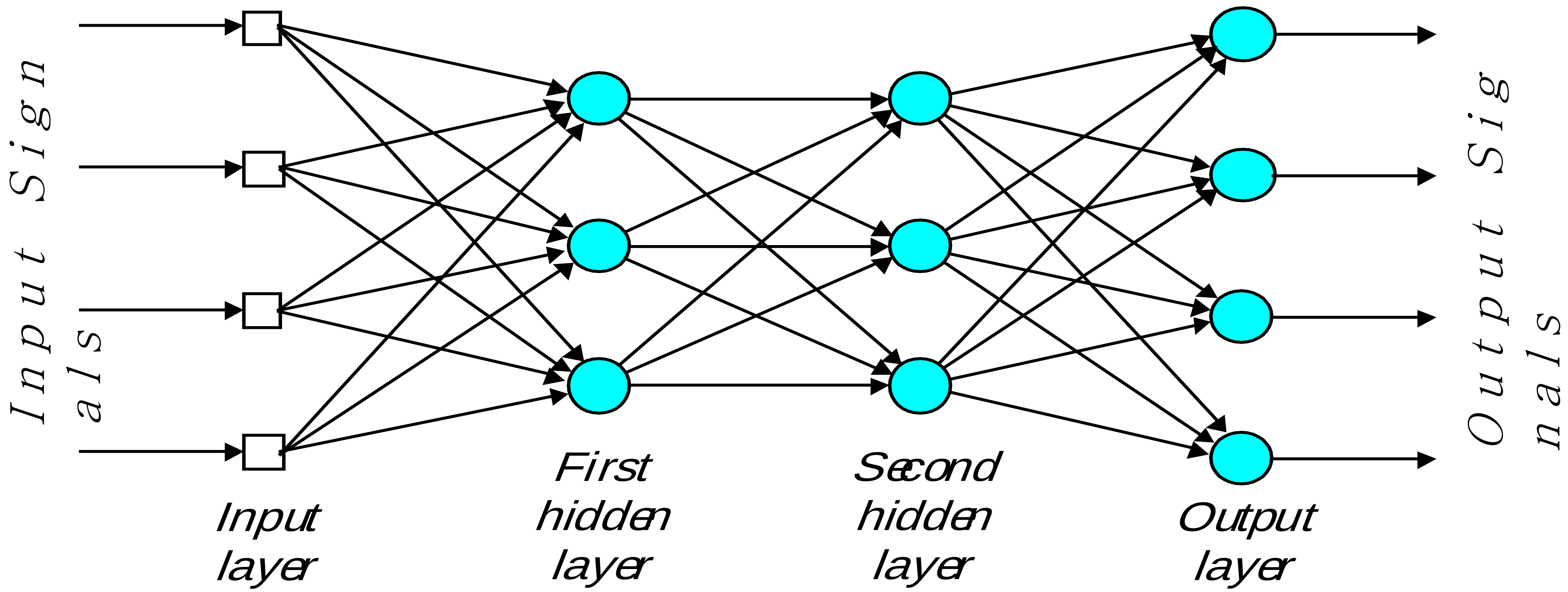
Multilayer Neural Network

- **Multilayer Perceptron (MLP)**
- **Hopfield**
- **Bidirectional associative memory**

Multilayer Perceptron (MLP)

- A multilayer perceptron is a feedforward neural network with one or more hidden layers.
- The network consists of an **input layer** of source neurons, at least one middle or **hidden layer** of computational neurons, and an **output layer** of computational neurons.
- The input signals are propagated in a forward direction on a layer-by-layer basis.

An Example of MLP with two hidden layers



What does the middle layer hide?

- A hidden layer "hides" its desired output. Neurons in the hidden layer cannot be observed through the input/output behaviour of the network. There is no obvious way to know what the desired output of the hidden layer should be. This is why ANN is considered a black box.
- Commercial ANNs incorporate three and sometimes four layers, including one or two hidden layers. Each layer can contain from 10 to 1000 neurons, depending on the learning algorithm employed. Experimental neural networks may have five or even six layers, including three or four hidden layers, and utilise millions of neurons.

How to determine an optimum MLP inputs?

- The number of input, hidden and output neurons must be determined in order to produce an optimum ANN structure.
- The number of ANN inputs depend on the number of features that is available for a problem.
  - Insufficient features may degrade the performance of an ANN.
  - Too many features may cause correlation and hence degrade an ANN’s performance due to confusion. To avoid correlation, some statistical methods such as Principal Components Analysis (PCA) and Discriminant Analysis (DA) can be employed to eliminate correlated inputs before an ANN training process.

How to determine an optimum MLP hiddens?

- Too few hidden neurons results in a MLP which is not capable of solving a problem. Too many causes the problem of data *overfitting* causing the MLP to be incapable of generalizing.
- The optimum number of hidden neurons is determined using either:
  - network growing - start with 1 hidden neuron and increase the number until there is no further improvement in performance.
  - network pruning approach – start with a sufficiently large number of neurons and reduce the number until a superior ANN performance is achieved.
- If there are too many neurons in a hidden layer, extra hidden layer may be introduced into the MLP.

How to determine an optimum MLP outputs?

- The optimum number of output neurons is problem dependent.
- For example: Classification of star fruit based on “Unripe”, Under ripe”, “Ripe” and “Over ripe”, either 2 or 4 output neurons can be used:
  - 2 output neurons can be representing binary value output cases of:

0 0 – for “unripe”	0 1 – for “under ripe”
1 0 – for “ripe”	1 1 – for “over ripe”

  - 4 output neurons can be representing binary bit output cases of:

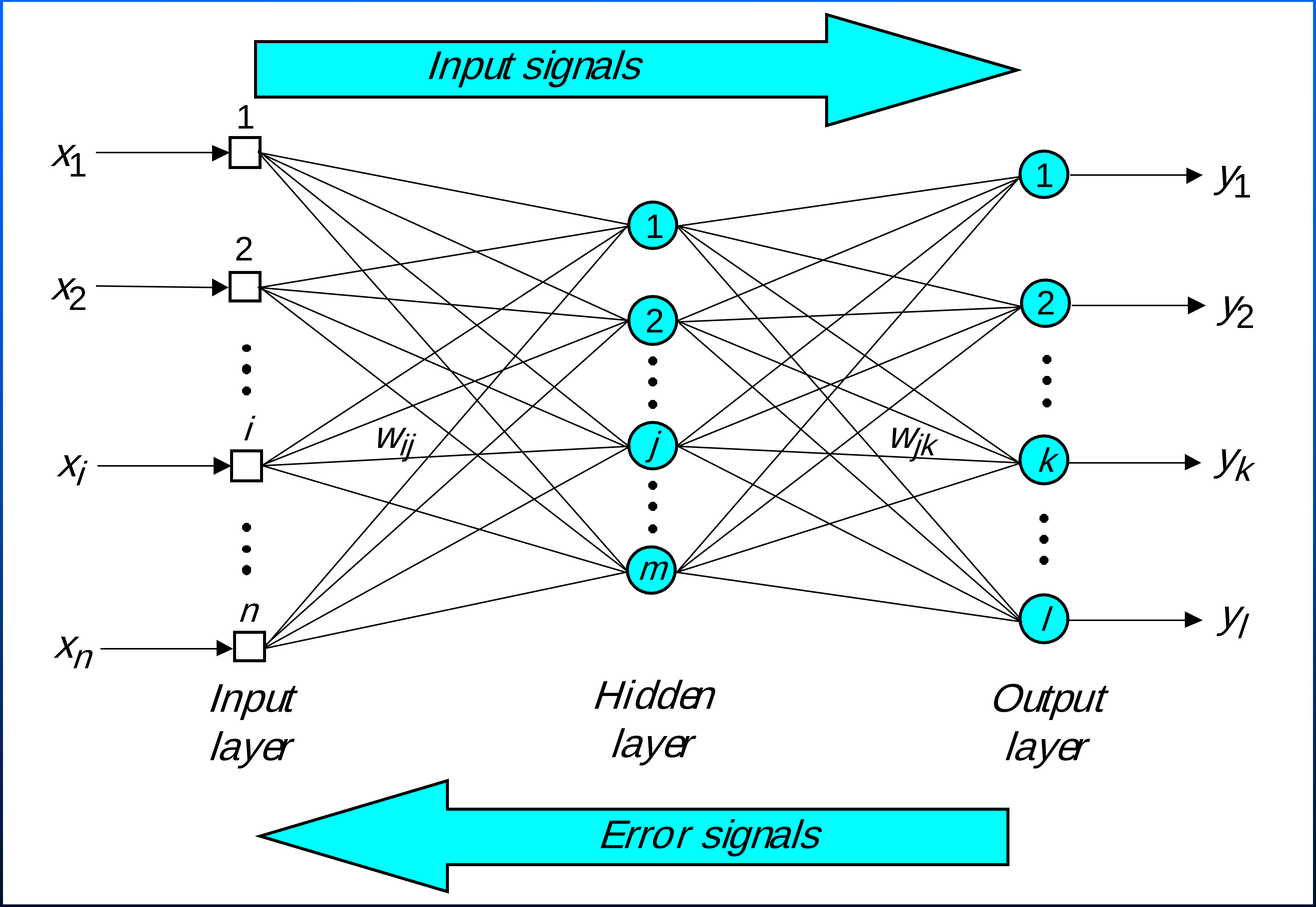
0 0 0 1 – for “unripe”	0 0 1 0 – for “under ripe”
0 1 0 0 – for “ripe”	1 0 0 0 – for “over ripe”

How MLP Learns

- Learning in a MLP NN proceeds the same way as for a perceptron.
- MLP is a backpropagation NN because it feeds back its error for updating or tuning of weights.
- Firstly, a training set of input patterns is presented to the network. At the same time, the desired or target output is retained for error calculation.
- The MLP computes its output and compares it with the desired output. If there is an error, i.e. difference between actual and desired output patterns, the MLP weights are adjusted based on a function to reduce the error.

- In short, a back-propagation neural network learning algorithm has two phases.
- First, a training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output pattern is generated by the output layer.
- If this pattern is different from the desired output, an error is calculated and then propagated backwards through the network from the output layer to the input layer. The weights are modified as the error is propagated.

Three-layer back-propagation neural network



The back-propagation training algorithm

Step 1: Initialisation

Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range:

$$\left(-\frac{2.4}{F_i}, \; +\frac{2.4}{F_i} \right)$$

where F_i is the total number of inputs of neuron i in the network. The weight initialisation is done on a neuron-by-neuron basis.

Step 2: Activation

Activate the back-propagation neural network by applying inputs $x_1(p), x_2(p), \ldots, x_n(p)$ and desired outputs $y_{d,1}(p), y_{d,2}(p), \ldots, y_{d,n}(p)$.

(a) Calculate the actual outputs of the neurons in the hidden layer:

$$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \cdot w_{ij}(p) - \theta_j\right]$$

where n is the number of inputs of neuron j in the hidden layer, and *sigmoid* is the *sigmoid* activation function.

Step 2 : Activation (continued)

(*b*) Calculate the actual outputs of the neurons in the output layer:

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \cdot w_{jk}(p) - \theta_k\right]$$

where *m* is the number of inputs of neuron *k* in the output layer.

Step 3: Weight training

Update the weights in the back-propagation network propagating backward the errors associated with output neurons.

(*a*) Calculate the error gradient for the neurons in the output layer:

$$\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p)$$

where $e_k(p) = y_{d,k}(p) - y_k(p)$

Calculate the weight corrections:

$$\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)$$

Update the weights at the output neurons:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

Step 3: Weight training (continued)

(*b*) Calculate the error gradient for the neurons in the hidden layer:

$$\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p)\, w_{jk}(p)$$

Calculate the weight corrections:

$$\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p)$$

Update the weights at the hidden neurons:

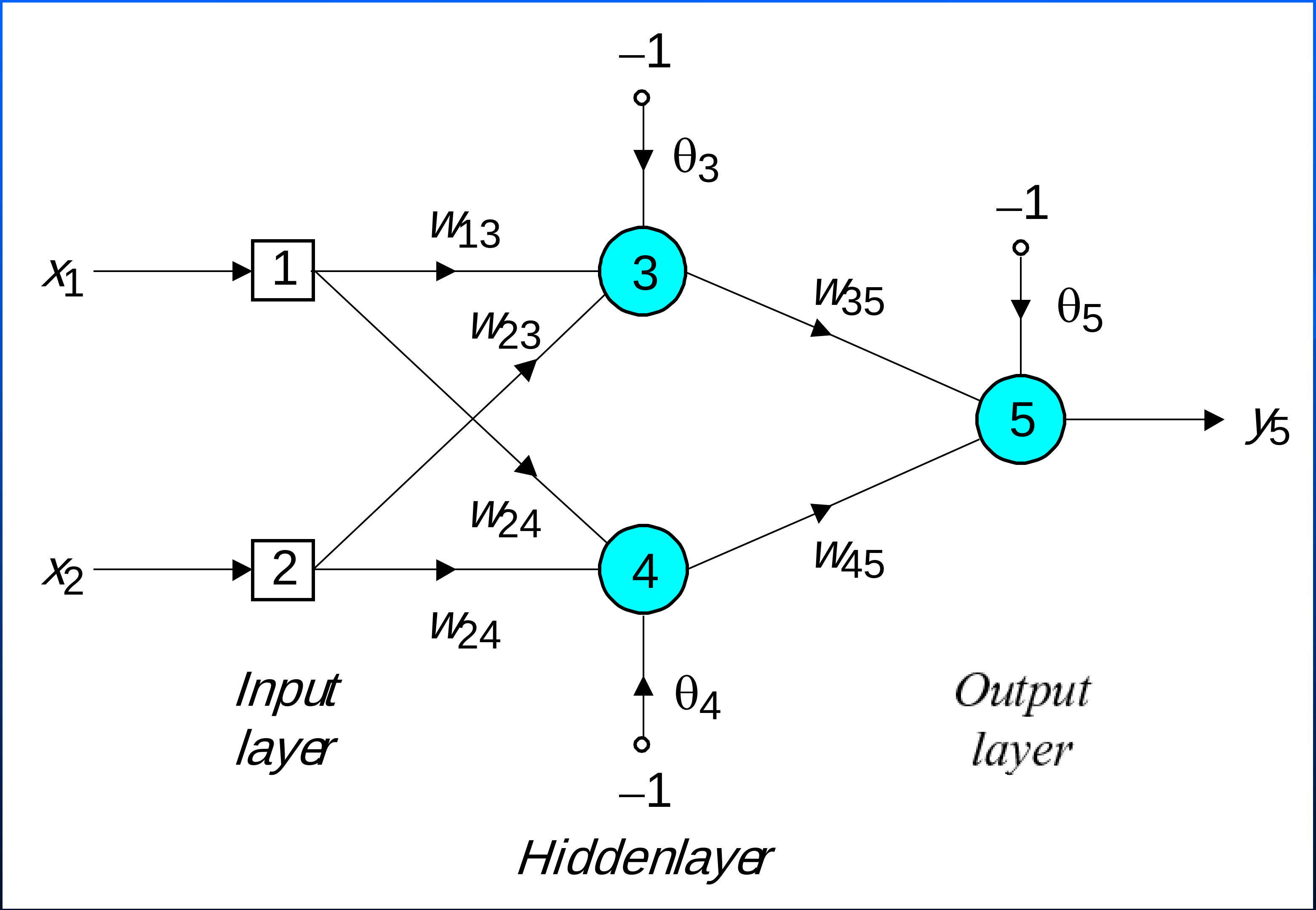
$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

Step 4: Iteration

Increase iteration *p* by one, go back to *Step 2* and repeat the process until the selected error criterion is satisfied.

As an example, we may consider the three-layer back-propagation network. Suppose that the network is required to perform logical operation *Exclusive-OR*. Recall that a single-layer perceptron could not do this operation. Now we will apply the three-layer net.

Three-layer network for solving the Exclusive-OR (XOR) operation



- The effect of the threshold applied to a neuron in the hidden or output layer is represented by its weight, θ, connected to a fixed input equal to –1.
- The initial weights and threshold levels are set randomly as follows:
  $w_{13} = 0.5$, $w_{14} = 0.9$, $w_{23} = 0.4$, $w_{24} = 1.0$, $w_{35} = -1.2$, $w_{45} = 1.1$, $\theta_3 = 0.8$, $\theta_4 = -0.1$ and $\theta_5 = 0.3$.

- We consider a training set where inputs x_1 and x_2 are equal to 1 and desired output $y_{d,5}$ is 0. The actual outputs of neurons 3 and 4 in the hidden layer are calculated as

$$y_3 = sigmoid\,(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1 \cdot 0.5 + 1 \cdot 0.4 - 1 \cdot 0.8)}\right] = 0.5250$$

$$y_4 = sigmoid\,(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1 \cdot 0.9 + 1 \cdot 1.0 + 1 \cdot 0.1)}\right] = 0.8808$$

- Now the actual output of neuron 5 in the output layer is determined as:

$$y_5 = sigmoid\,(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/\left[1 + e^{-(-0.5250 \cdot 1.2 + 0.8808 \cdot 1.1 - 1 \cdot 0.3)}\right] = 0.5097$$

- Thus, the following error is obtained:

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$

- The next step is weight training. To update the weights and threshold levels in our network, we propagate the error, *e*, from the output layer backward to the input layer.

- First, we calculate the error gradient for neuron 5 in the output layer:

$$\delta_5 = y_5\,(1 - y_5)\,e = 0.5097 \cdot (1 - 0.5097) \cdot (-0.5097) = -0.1274$$

- Then we determine the weight corrections assuming that the learning rate parameter, α, is equal to 0.1:

$$\Delta w_{35} = \alpha \cdot y_3 \cdot \delta_5 = 0.1 \cdot 0.5250 \cdot (-0.1274) = -0.0067$$
$$\Delta w_{45} = \alpha \cdot y_4 \cdot \delta_5 = 0.1 \cdot 0.8808 \cdot (-0.1274) = -0.0112$$
$$\Delta \theta_5 = \alpha \cdot (-1) \cdot \delta_5 = 0.1 \cdot (-1) \cdot (-0.1274) = -0.0127$$

- Next we calculate the error gradients for neurons 3 and 4 in the hidden layer:

$$\delta_3 = y_3(1 - y_3) \cdot \delta_5 \cdot w_{35} = 0.5250 \cdot (1 - 0.5250) \cdot (-0.1274) \cdot (-1.2) = 0.0381$$

$$\delta_4 = y_4(1 - y_4) \cdot \delta_5 \cdot w_{45} = 0.8808 \cdot (1 - 0.8808) \cdot (-0.1274) \cdot 1.1 = -0.0147$$

- We then determine the weight corrections:

$$\Delta w_{13} = \alpha \cdot x_1 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$
$$\Delta w_{23} = \alpha \cdot x_2 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$
$$\Delta \theta_3 = \quad \cdot (-1) \cdot \quad _3 = 0.1 \cdot (-1) \cdot 0.0381 = -0.0038$$
$$\Delta w_{14} = \quad \cdot x_1 \cdot \quad _4 = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$
$$\Delta w_{24} = \quad \cdot x_2 \cdot \quad _4 = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$
$$\Delta \theta_4 = \quad \cdot (-1) \cdot \quad _4 = 0.1 \cdot (-1) \cdot (-0.0147) = 0.0015$$

- At last, we update all weights and threshold:

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$

$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$

$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$

$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$

$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$

$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$

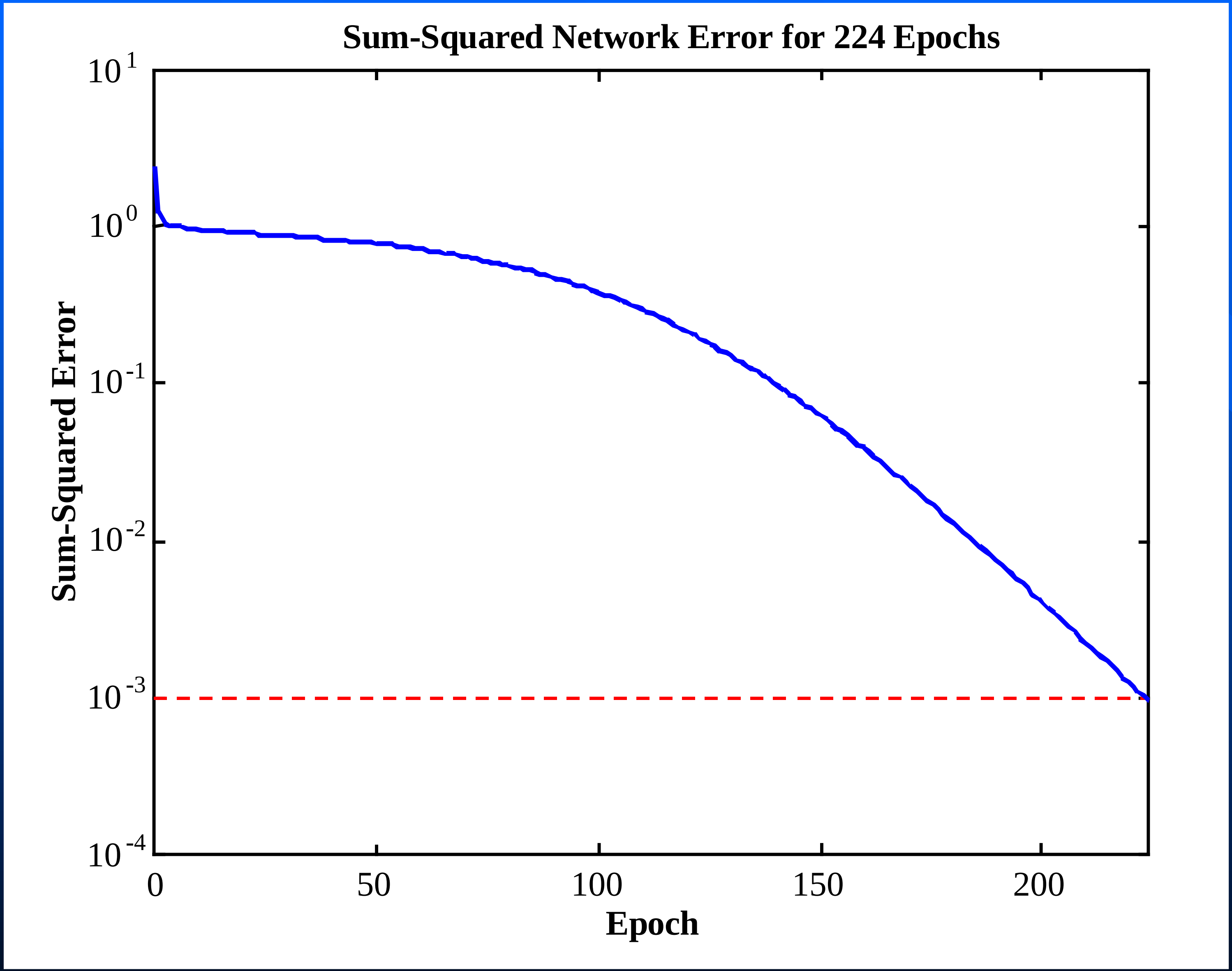
$$\theta_3 = \theta_3 + \Delta\theta_3 = 0.8 - 0.0038 = 0.7962$$

$$\theta_4 = \theta_4 + \Delta\theta_4 = -0.1 + 0.0015 = -0.0985$$

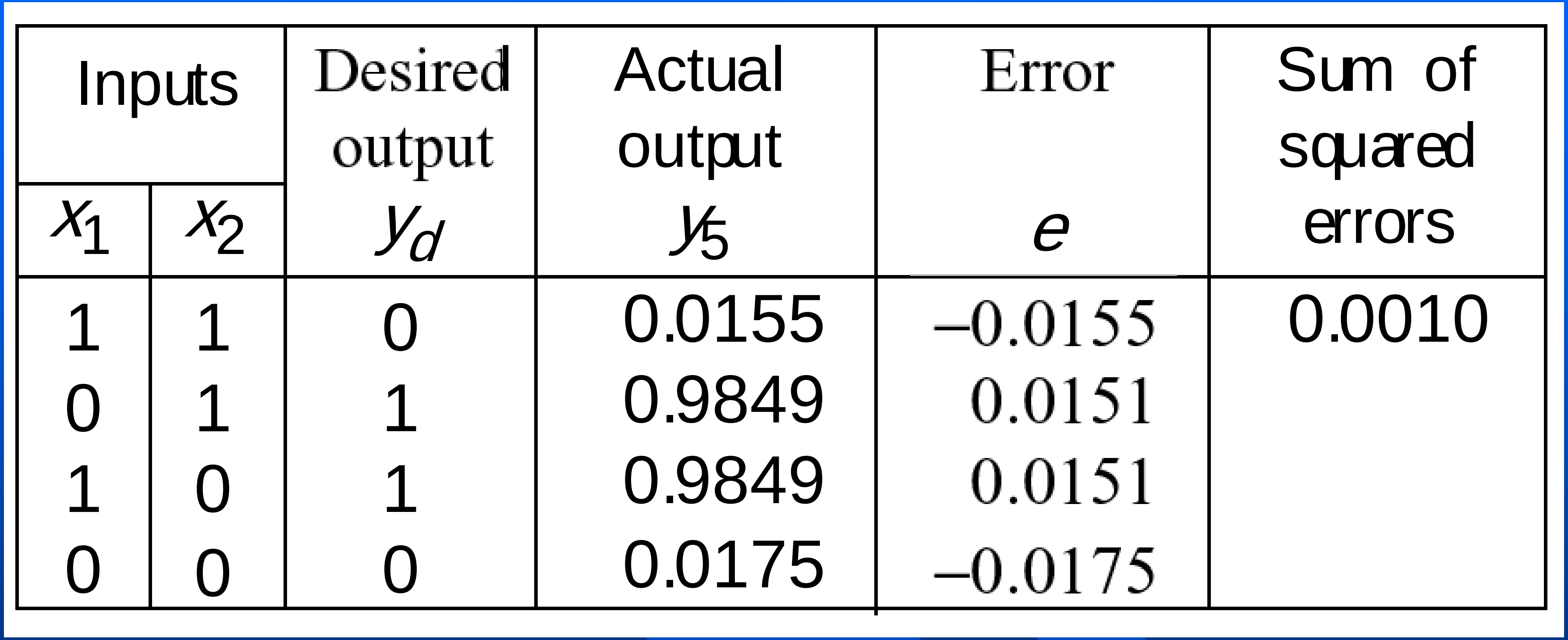
$$\theta_5 = \theta_5 + \Delta\theta_5 = 0.3 + 0.0127 = 0.3127$$

- The new weights are used in the next iteration or epoch.

Learning curve for operation *Exclusive-OR*

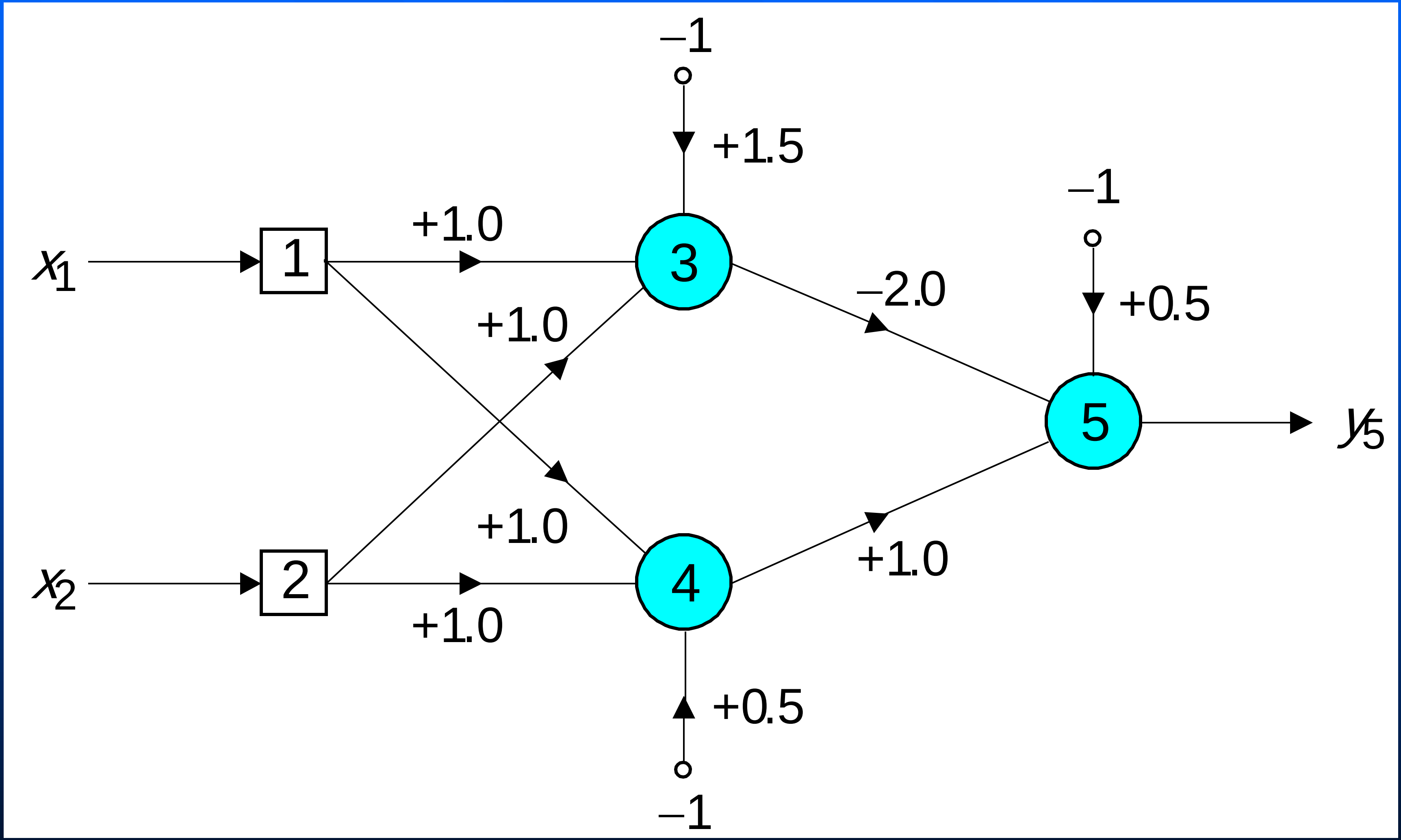


Final results of three-layer network learning

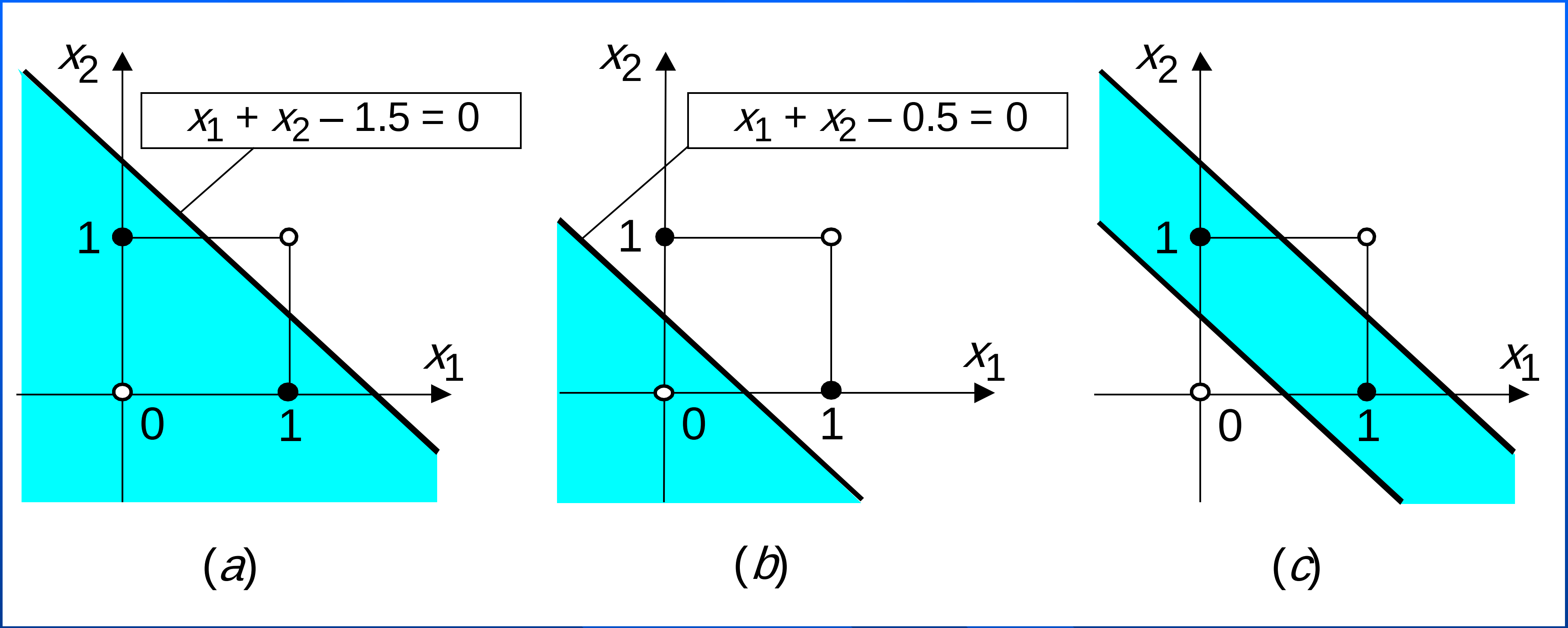


Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	–0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	–0.0175	

Network represented by McCulloch-Pitts model for solving the *Exclusive-OR* operation



Decision boundaries



(a) Decision boundary constructed by hidden neuron 3;
(*b*) Decision boundary constructed by hidden neuron 4;
(*c*) Decision boundaries constructed by the complete three-layer network

Accelerated learning in multilayer neural networks

- A multilayer network learns much faster when the sigmoidal activation function is represented by a **hyperbolic tangent**:

$$Y^{tanh} = \frac{2a}{1 + e^{-bX}} - a$$

where *a* and *b* are constants.

Suitable values for *a* and *b* are:
a = 1.716 and *b* = 0.667

- We also can accelerate training by including a **momentum term** in the delta rule:

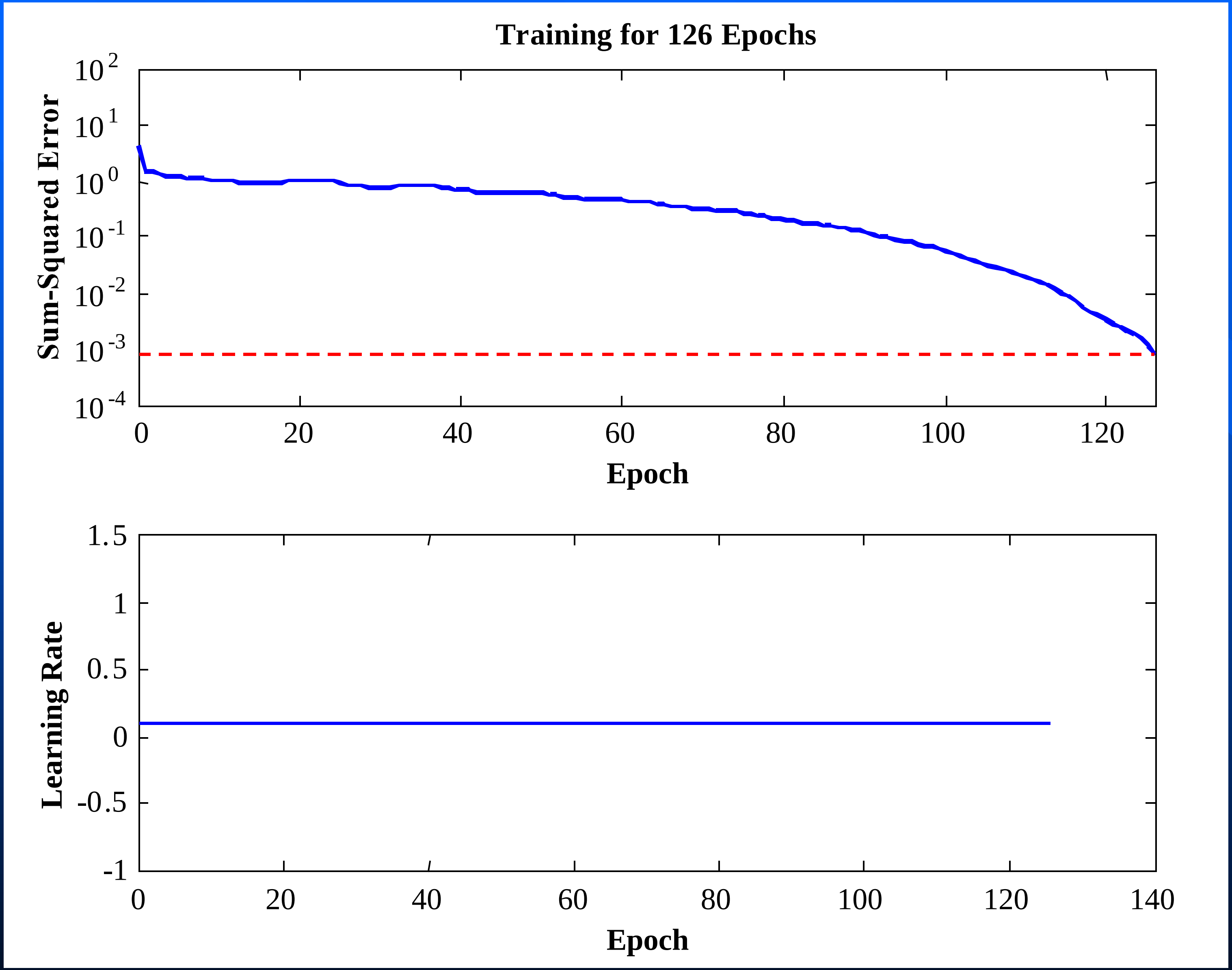
$$\Delta w_{jk}(p) = \beta \cdot \Delta w_{jk}(p-1) + \alpha \cdot y_j(p) \cdot \delta_k(p)$$

where β is a positive number ($0 \leq \beta < 1$) called the **momentum constant**. Typically, the momentum constant is set to 0.95.

This equation is called the **generalised delta rule**.

However, it does not guarantee convergence or achievement of global minimum.

Learning with momentum for operation *Exclusive-OR*



Learning with adaptive learning rate

To accelerate the convergence and yet avoid the danger of instability, we can apply two heuristics:

Heuristic 1

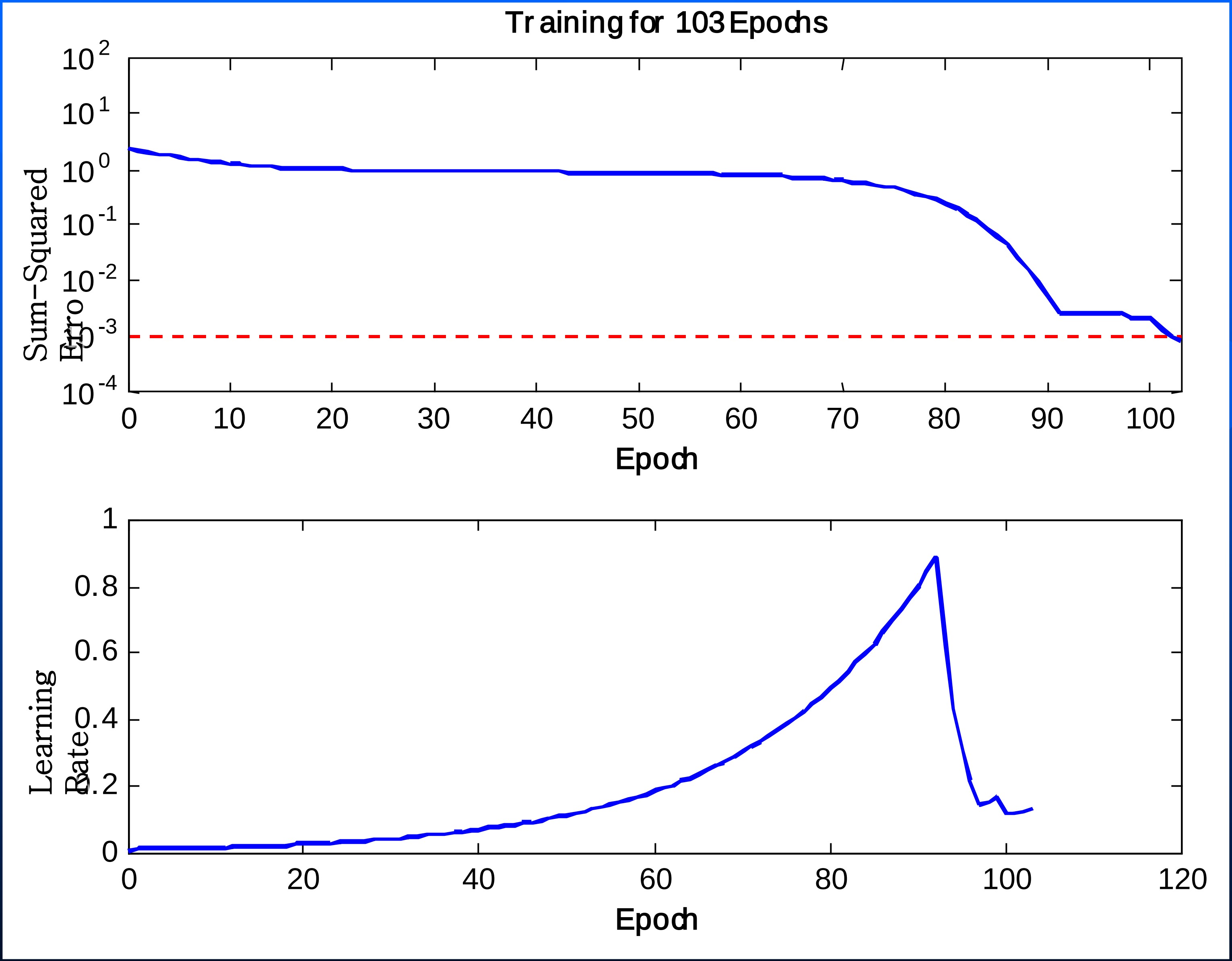
If the change of the sum of squared errors has the same algebraic sign for several consequent epochs, then the learning rate parameter, α, should be increased.

Heuristic 2

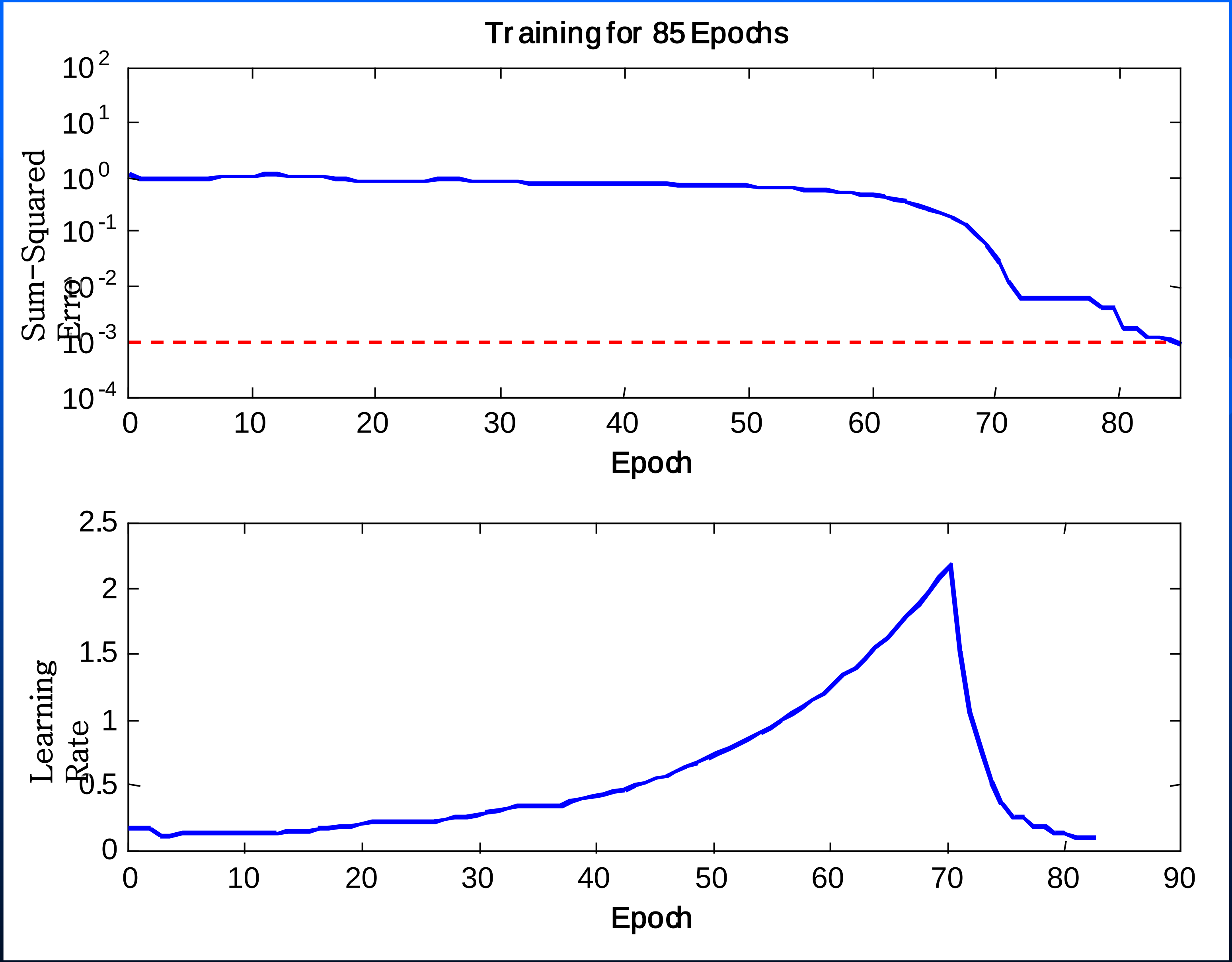
If the algebraic sign of the change of the sum of squared errors alternates for several consequent epochs, then the learning rate parameter, α, should be decreased.

- Adapting the learning rate requires some changes in the back-propagation algorithm.
- If the sum of squared errors at the current epoch exceeds the previous value by more than a predefined ratio (typically 1.04), the learning rate parameter is decreased (typically by multiplying by 0.7) and new weights and thresholds are calculated.
- If the error is less than the previous one, the learning rate is increased (typically by multiplying by 1.05).

Learning with adaptive learning rate



Learning with momentum and adaptive learning rate



Types of MLP Learning Algorithm

- Gradient Descent (GD) – slow due to small learning rate
- GD + momentum – fast but convergence not guaranteed
- GD + adaptive learning rate – ”
- GD + Newton-Raphson – good convergence
- Levenberg Marquardt (LM) – consistently good performance for classification tasks
- Scaled Conjugate Gradient (SCG) – fast but convergence not guaranteed
- Resillient BP – problem dependent, performance not consistent.
- Bayesian Regularization (BR) – slow but increased performance. Usually better than LM.

Limitations of MLP

- MLPs are universal approximators. They can solve most complex problems. However, they are limited in certain aspects.
- MLPs are static NN. They do not have memory to relate the previous information with the current.
- To emulate the human memory's associative characteristics we need a different type of network: a recurrent neural network.
- A recurrent neural network has feedback loops from its outputs to its inputs. The presence of such loops has a profound impact on the learning capability of the network.