Backpropagation Algorithm

Back Propagation (Gradient computation)

Phase 1: Propagation
Each propagation involves the following steps:

1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.

2. Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas of all output and hidden neurons.

Back Propagation (Gradient computation)

Phase 2: Weight update
For each weight-synapse follow the following steps:

1. Multiply its output delta and input activation to get the gradient of the weight.
2. Subtract a ratio (percentage) of the gradient from the weight.

The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.
Repeat phase 1 and 2 until the performance of the network is satisfactory.

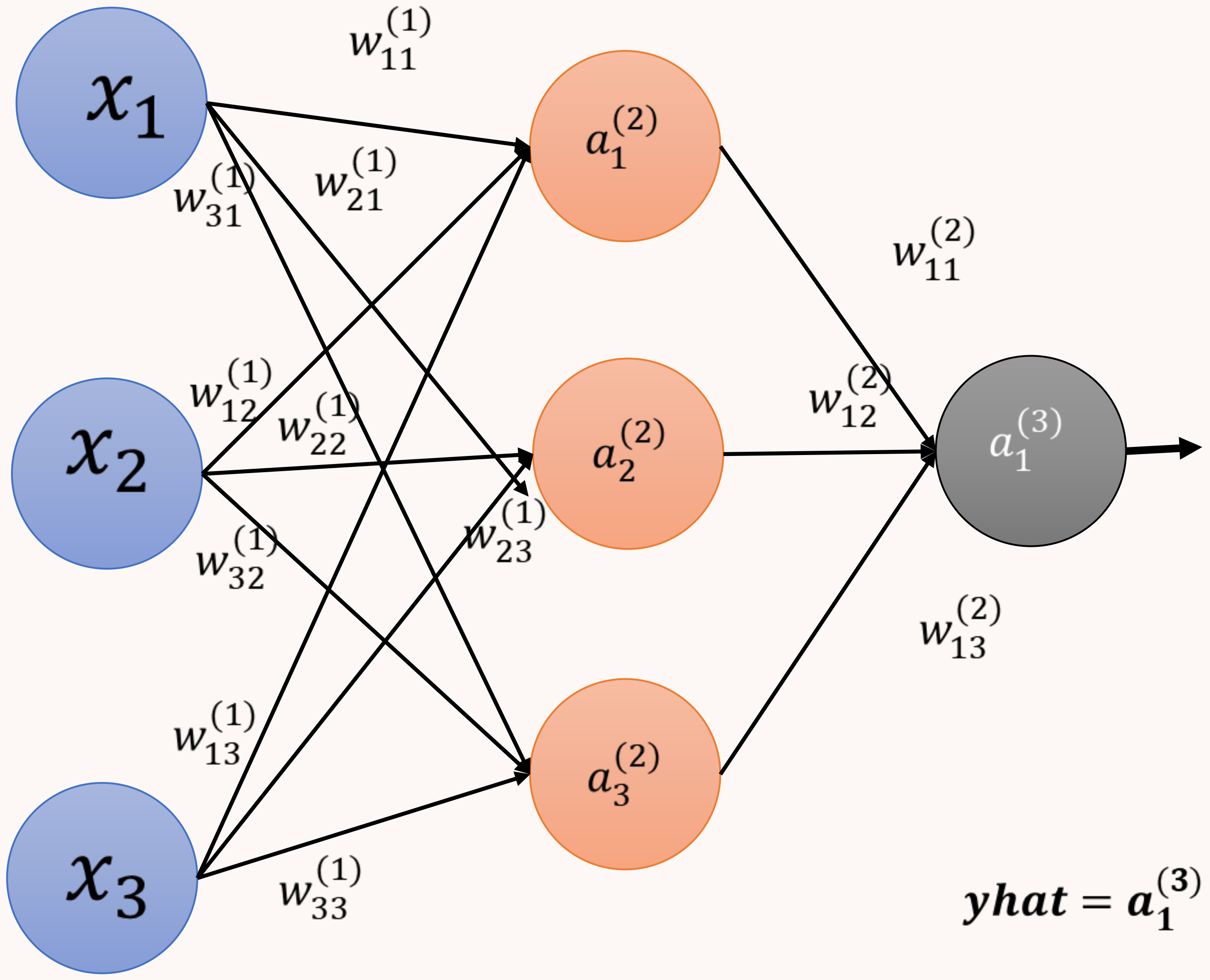
Back Propagation (Gradient computation)

Phase 2: Weight update
For each weight-synapse follow the following steps:

1. Multiply its output delta and input activation to get the gradient of the weight.
2. Subtract a ratio (percentage) of the gradient from the weight.

The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.
Repeat phase 1 and 2 until the performance of the network is satisfactory.

Mathematical Modelling of Neural Network



$$a_1^{(2)} = g\{w_{10}^{(1)}x_0 + w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3\}$$

$$a_2^{(2)} = g\{w_{20}^{(1)}x_0 + w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3\}$$

$$a_3^{(2)} = g\{w_{30}^{(1)}x_0 + w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3\}$$

$$\boldsymbol{yhat = a_1^{(3)} = g\{w_{10}^{(2)}a_0^{(2)} + w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)}\}}$$

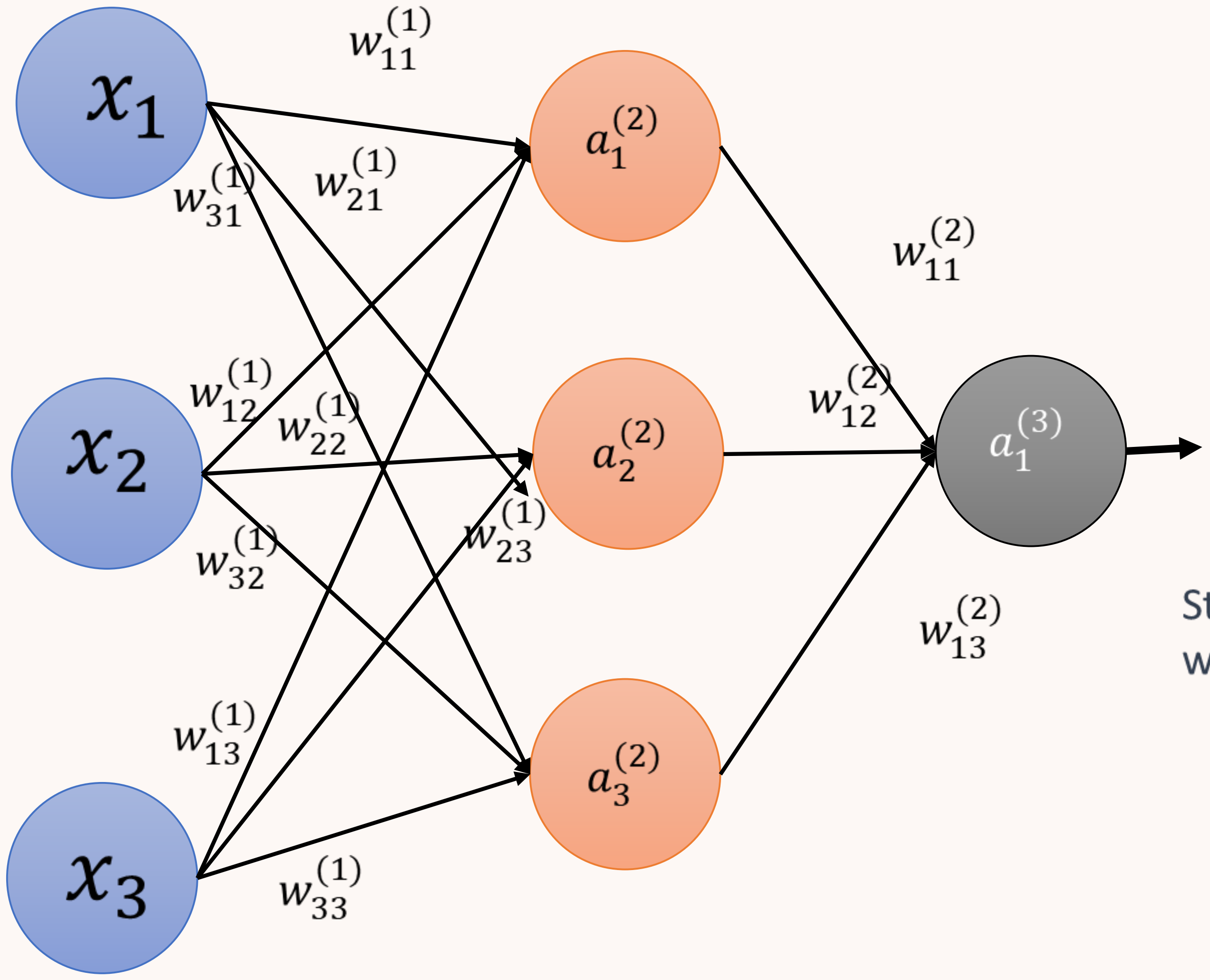
Back Propagation (Gradient computation)

Phase 2: Weight update
For each weight-synapse follow the following steps:

1. Multiply its output delta and input activation to get the gradient of the weight.
2. Subtract a ratio (percentage) of the gradient from the weight.

The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.
Repeat phase 1 and 2 until the performance of the network is satisfactory.

Mathematical Modelling of Neural Network



$$(Error)E = \frac{1}{2}(t - yhat)^2$$

E = mean square error
t = target value (Actual Value)
yhat = predicted output of the output neuron

Step 2 finding the derivative of the error with respect to weights

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = Gradient\ of\ error\ w.r.t.weight$$

Back Propagation (Gradient computation)

$$E = \frac{1}{2}(t-y)^2 = \frac{1}{2}(t - a_1^{(3)})^2$$

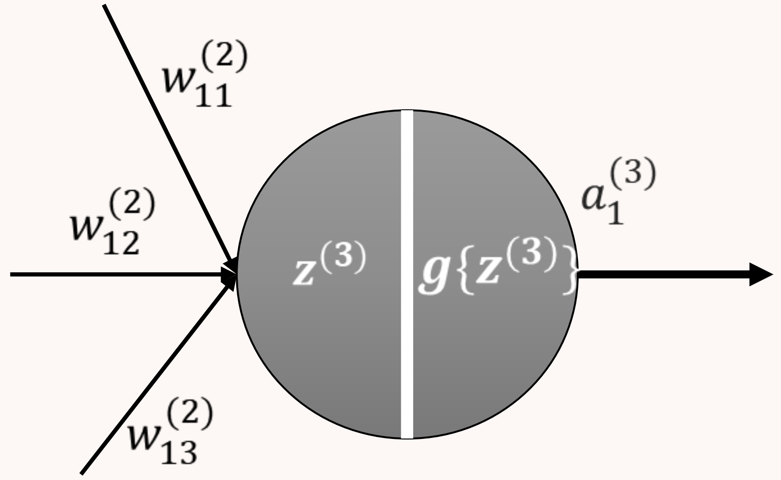
$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial \mathbf{z}^{(3)}} \frac{\partial \mathbf{z}^{(3)}}{\partial w_{ij}^{(l)}}$$

$$\frac{\partial \mathbf{z}^{(3)}}{\partial w_{ij}^{(l)}} = \frac{\partial}{\partial w_{ij}^{(l)}} \left(\sum_{k=1}^{n} (w_{kj}^{(l)} * a_k^{(l)}) \right) = a_k^{(l)}$$

$$\frac{\partial a_1^{(3)}}{\partial \mathbf{z}^{(3)}} = \frac{\partial g\{\mathbf{z}^{(3)}\}}{\partial \mathbf{z}^{(3)}} = g\{\mathbf{z}^{(3)}\}(1 - g\{\mathbf{z}^{(3)}\})$$

$$= a_1^{(3)}(1 - a_1^{(3)})$$

$$\frac{\partial E}{\partial a_1^{(3)}} = \frac{\partial}{\partial a_1^{(3)}} \frac{1}{2}(t-y)^2 = \frac{\partial}{\partial a_1^{(3)}} \frac{1}{2}(t - a_1^{(3)})^2 = (a_1^{(3)} - \text{t})$$



putting it altogether,

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = a_1^{(3)}(1 - a_1^{(3)}) * (a_1^{(3)} - \text{t}) * a_k^{(2)}$$

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta^{(\mathbf{3})} * a_k^{(2)}$$

where $\delta^{(\mathbf{3})} = a_1^{(3)}(1 - a_1^{(3)}) * (a_1^{(3)} - \text{t})$

Back Propagation Steps – Calculating delta 2

Step 1 Operate Feed Forward Network to find the y actual value of network and calculate the Error term on output neuron.

Step 2 Calculate δ for output neuron and hidden neurons, δ will not be calculated for input neuron

$$\delta^{(\mathbf{3})} = a_1^{(3)}(1\text{-}a_1^{(3)}) * \left(a_1^{(3)} - \mathrm{t}\right)$$

$$\delta^{(\mathbf{2})} = \delta^{(\mathbf{3})} w_{kj}^{(2)} a_1^{(2)}(1\text{-}a_1^{(2)})$$

The back-propagation training algorithm

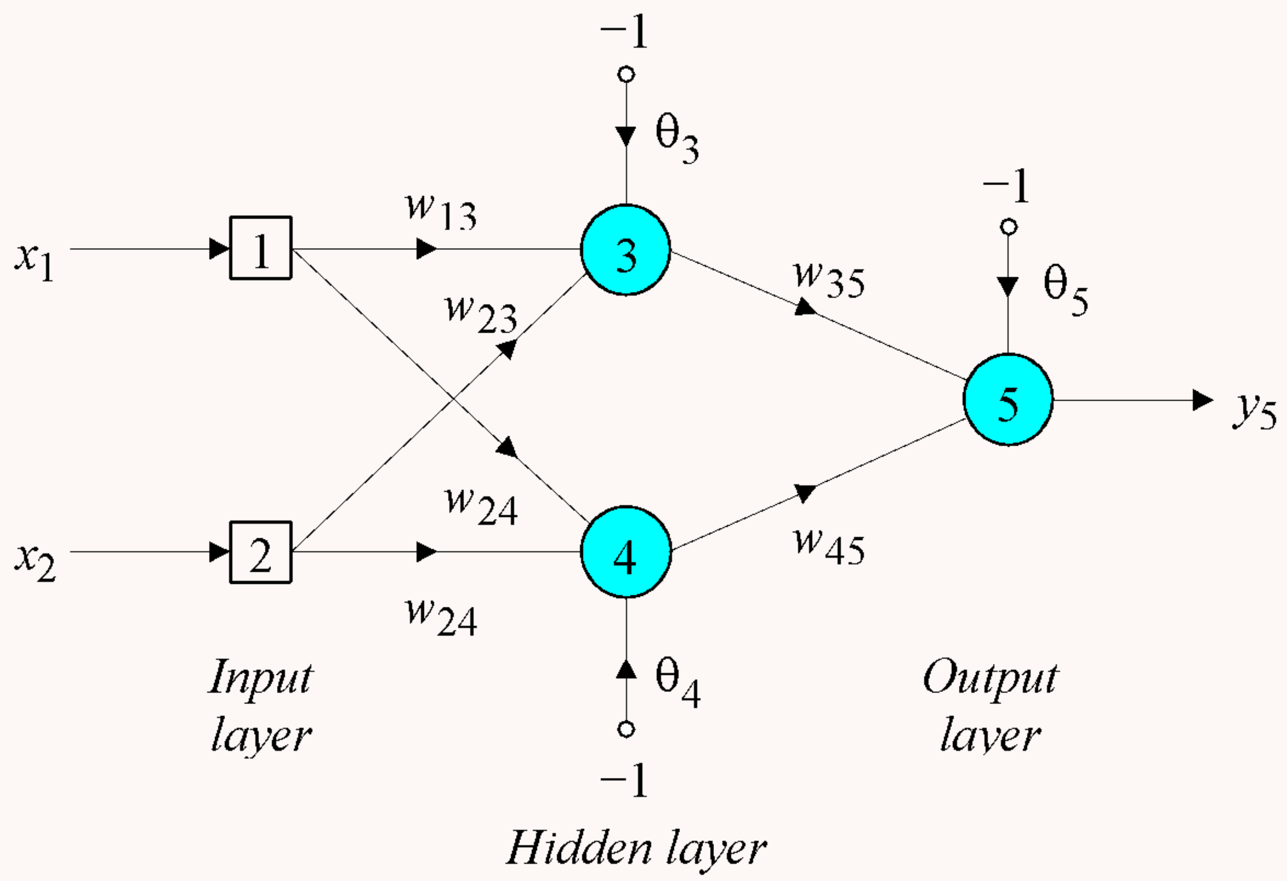
Step 1: Initialization

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

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

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

Example - Three-layer network



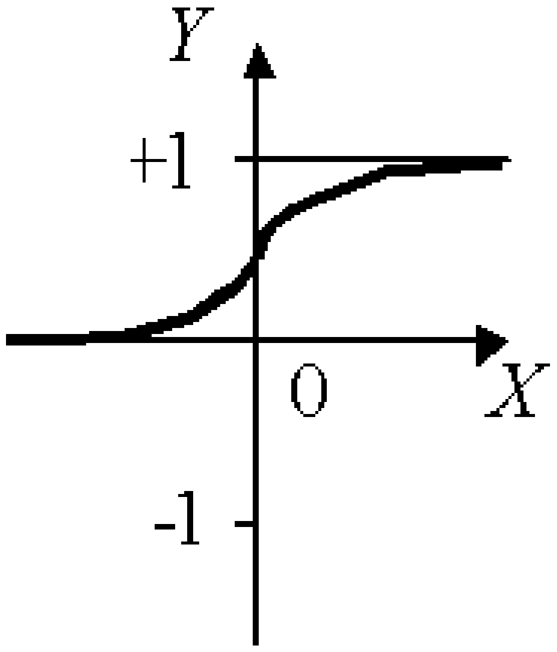
$w13 = 0.5$, $w14 = 0.9$,
$w23 = 0.4$, $w24 = 1.0$,
$w35 = -1.2$, $w45 = 1.1$,
$\theta 3 = 0.8$, $\theta 4 = -0.1$ and
$\theta 5 = 0.3$

Example Step 1

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 e.g., as follows:
w13 = 0.5, w14 = 0.9, w23 = 0.4, w24 = 1.0, w35 = −1.2, w45 = 1.1, θ3 = 0.8, θ4 = −0.1 and θ5 = 0.3.

Assuming the sigmoid activation Function



$$Y^{sigmoid} = \frac{1}{1+e^{-X}}$$

Example Step 2 - Feed Forward

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:

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

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

Example Step 2

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.

If the sigmoid activation function is used the output of the hidden layer is

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

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

And the actual output of neuron 5 in the output layer is

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

And the error is

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

What learning law applies in a multilayer neural network?

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

Step 3: Weight training output layer

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

(a) Calculate the error

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

and then the error gradient for the neurons in the output layer:

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

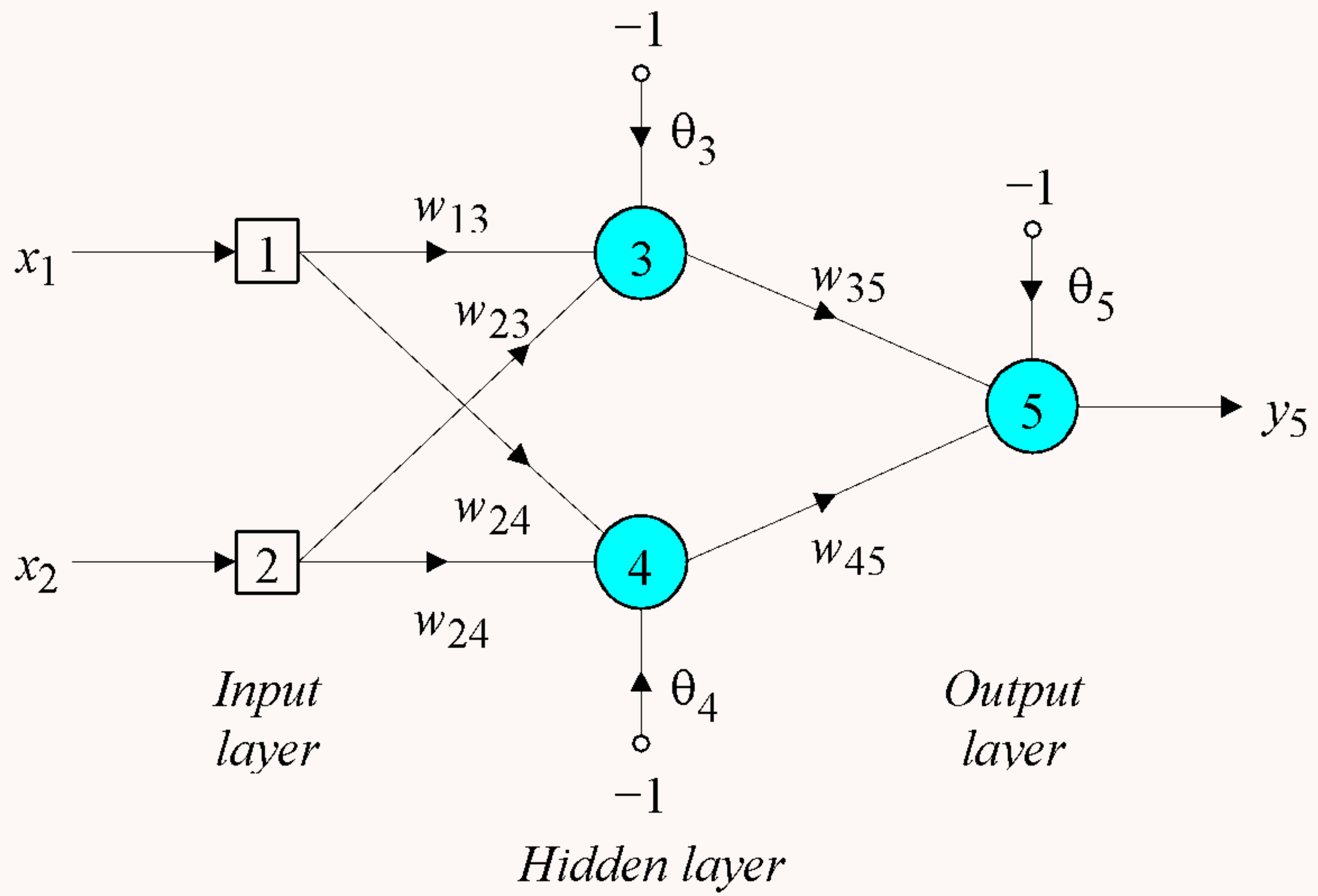
Then the weight corrections:

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

Then the new weights at the output neurons:

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

Three-layer network for solving the Exclusive-OR operation



- The error gradient for neuron 5 in the output layer:

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

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

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

Step 3: Weight training hidden layer

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

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

Calculate the weight corrections:

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

Update the weights at the hidden neurons:

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

- The error gradients for neurons 3 and 4 in the hidden layer:

$$\boxed{\begin{aligned}\delta_3 &= y_3(1-y_3)\cdot\delta_5\cdot w_{35} = 0.5250\cdot(1-0.5250)\cdot(-0.1274)\cdot(-1.2) = 0.0381\\ \delta_4 &= y_4(1-y_4)\cdot\delta_5\cdot w_{45} = 0.8808\cdot(1-0.8808)\cdot(-0.127\,4)\cdot 1.1 = -0.0147\end{aligned}}$$

- Determine the weight corrections:

$$\boxed{\begin{aligned}\Delta w_{13} &= \alpha\cdot x_1\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038\\ \Delta w_{23} &= \alpha\cdot x_2\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038\\ \Delta\theta_3 &= \alpha\cdot(-1)\cdot\delta_3 = 0.1\cdot(-1)\cdot 0.0381 = -0.0038\\ \Delta w_{14} &= \alpha\cdot x_1\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015\\ \Delta w_{24} &= \alpha\cdot x_2\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015\\ \Delta\theta_4 &= \alpha\cdot(-1)\cdot\delta_4 = 0.1\cdot(-1)\cdot(-0.0147) = 0.0015\end{aligned}}$$

- At last, we update all weights and threshold:

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

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

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

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

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

$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.011 \quad = 1.0888$$

2

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

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

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

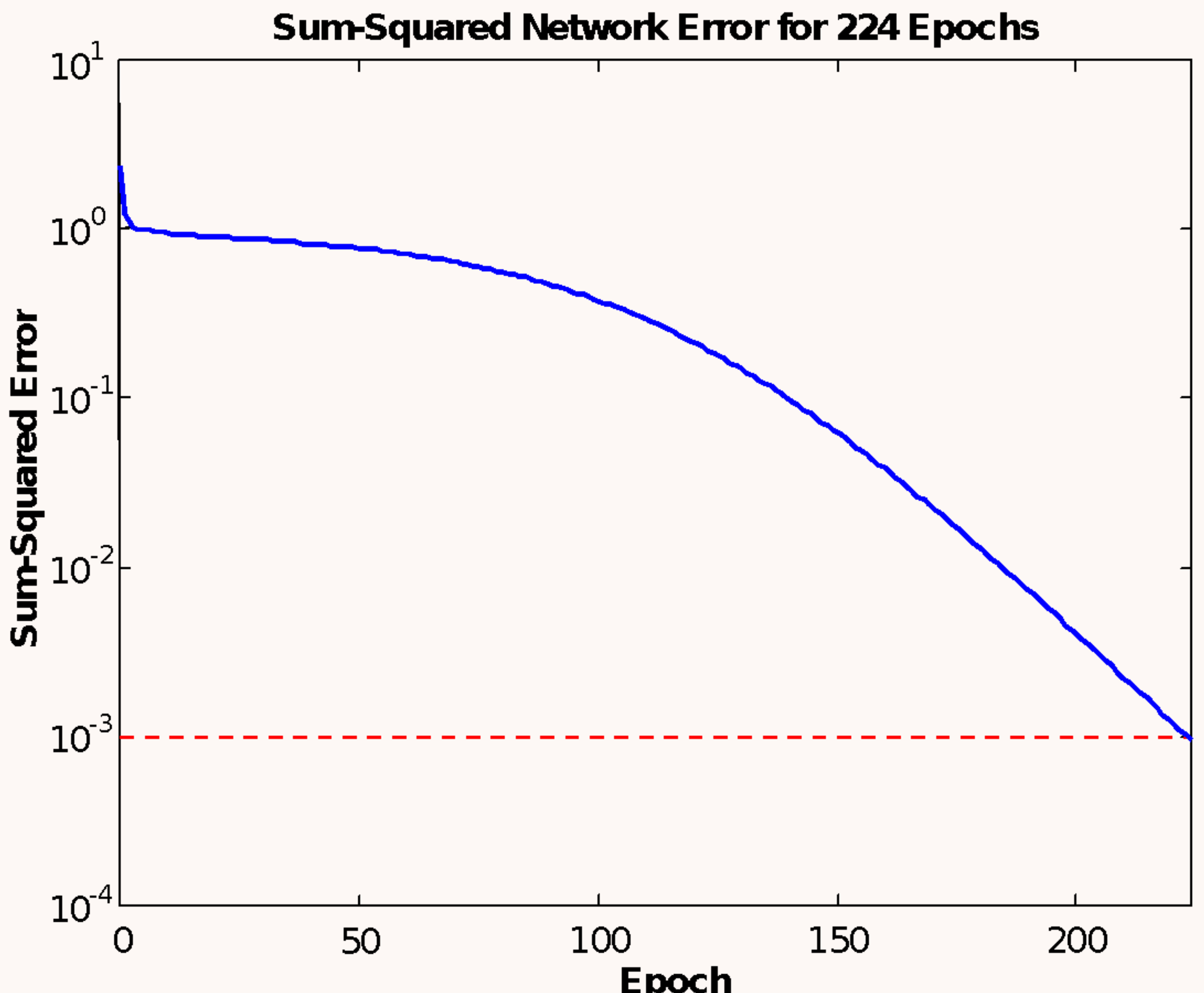
- The training process is repeated until the sum of squared errors is less than 0.001.

Step 4: Iteration

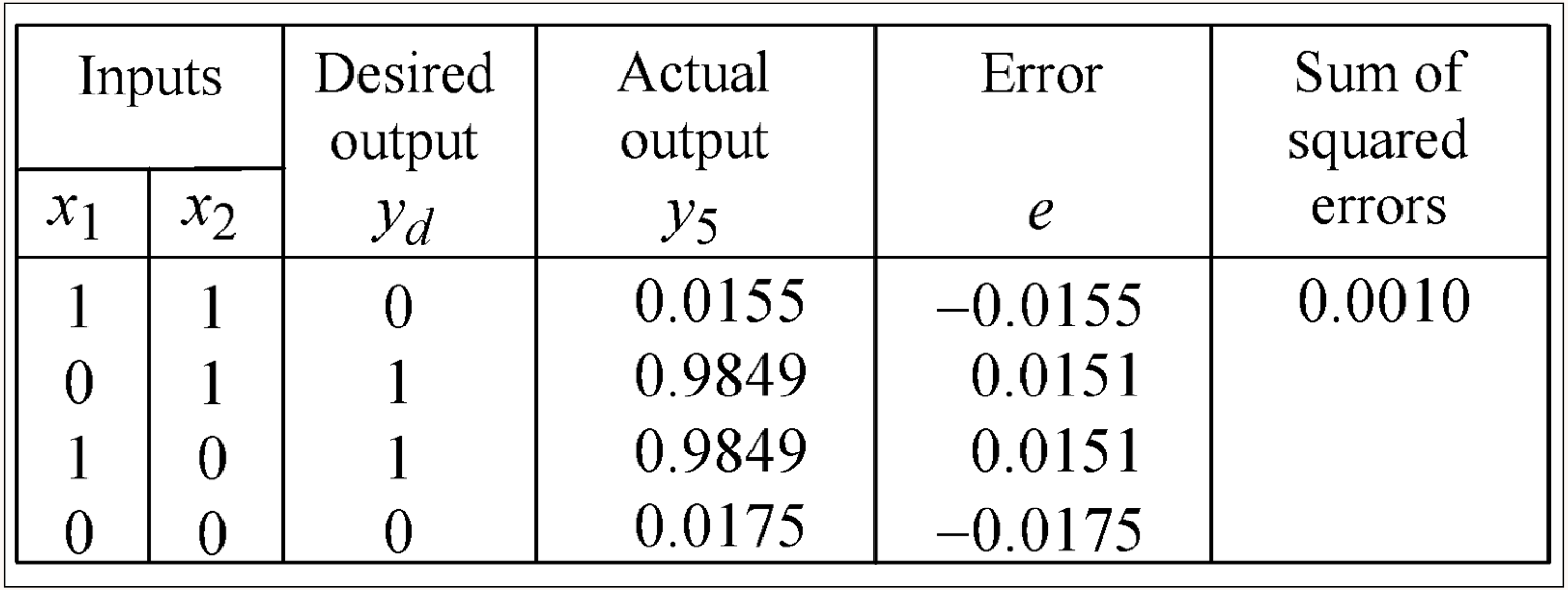
Increase iteration *p* by one, go back to *Step 2* and repeat the process until the selected error criterion is satisfied.

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.

Typical Learning Curve



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	

Accelerated learning in multilayer neural networks

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

$$\boxed{Y^{tan\,h} = \frac{2a}{1+e^{-bX}} - a}$$

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:

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

where β is a positive number (0 ≤ β • 1) called the **momentum constant**. Typically, the momentum constant is set to 0.95.

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

Learning with an adaptive learning rate

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

Heuristic

If the error is decreasing the learning rate α, should be increased.

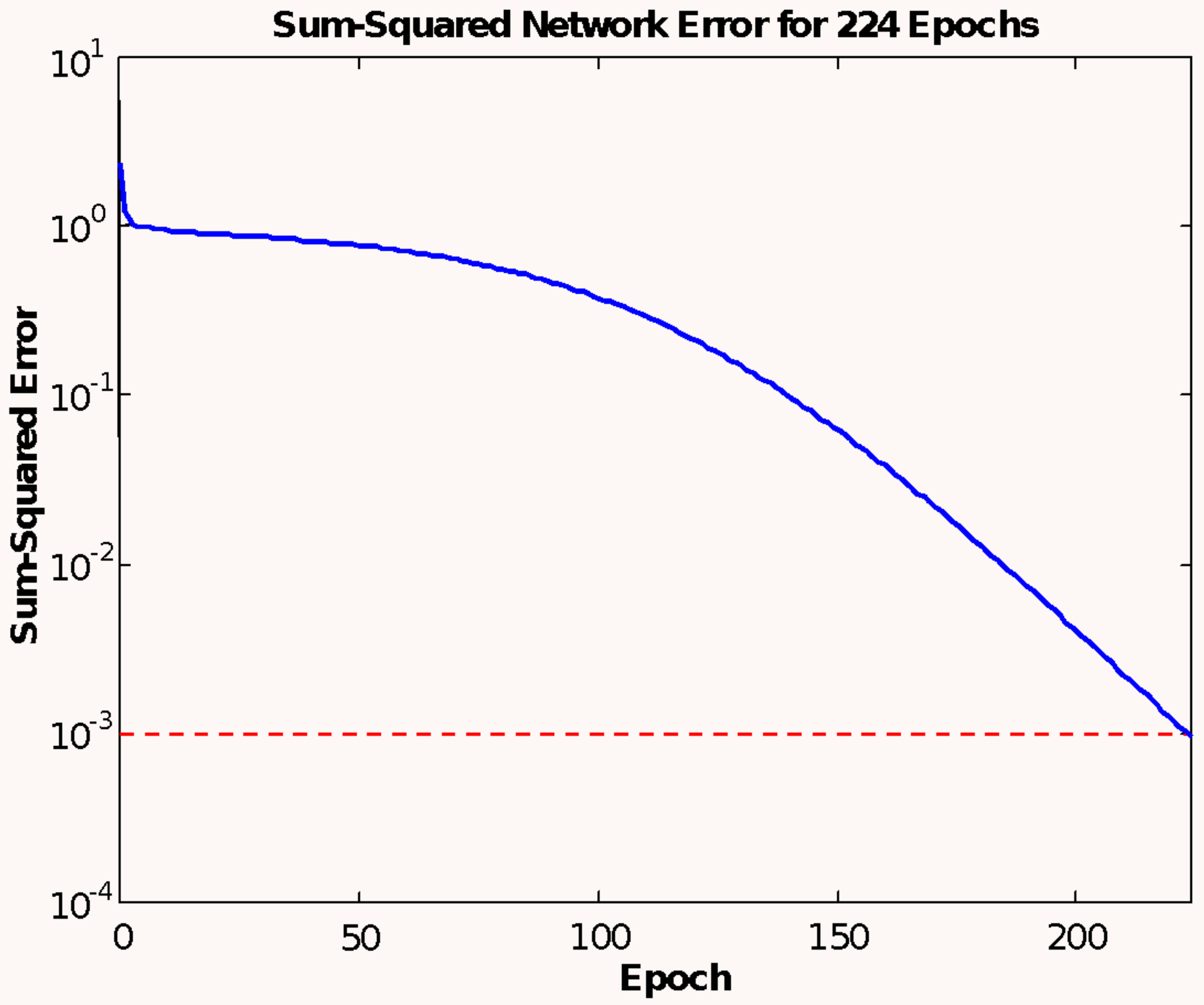
If the error is increasing or remaining constant the learning rate α, should be decreased.

- Adapting the learning rate requires some changes in the back-propagation algorithm.

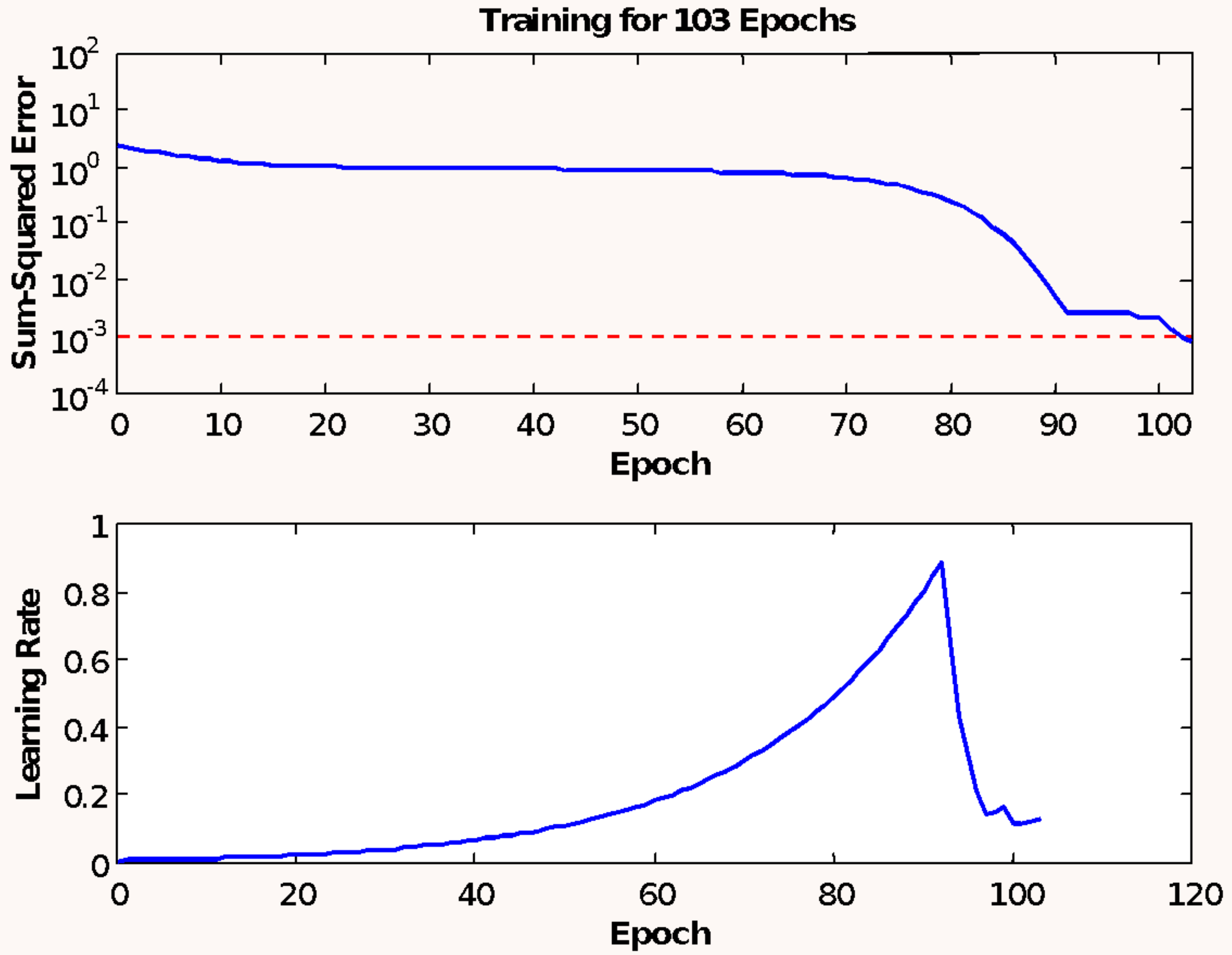
- If the sum of squared errors at the current epoch exceeds the previous value by more than a predefined ratio (typically 1.04), the learning rate parameter is decreased (typically by multiplying by 0.7) and new weights and thresholds are calculated.

- If the error is less than the previous one, the learning rate is increased (typically by multiplying by 1.05).

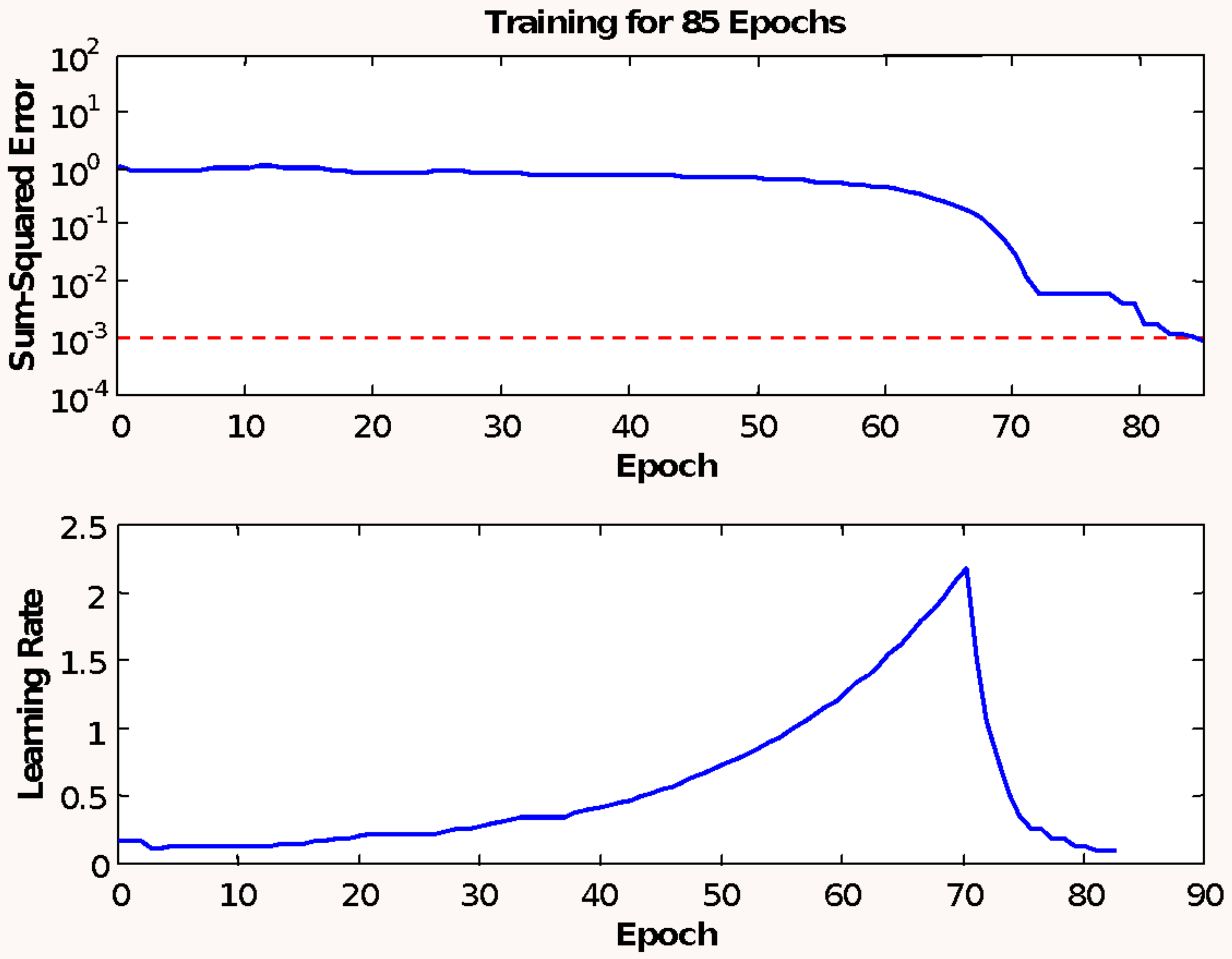
Typical Learning Curve



Typical learning with adaptive learning rate



Typical Learning with adaptive learning rate plus momentum



Thank You

ICASR