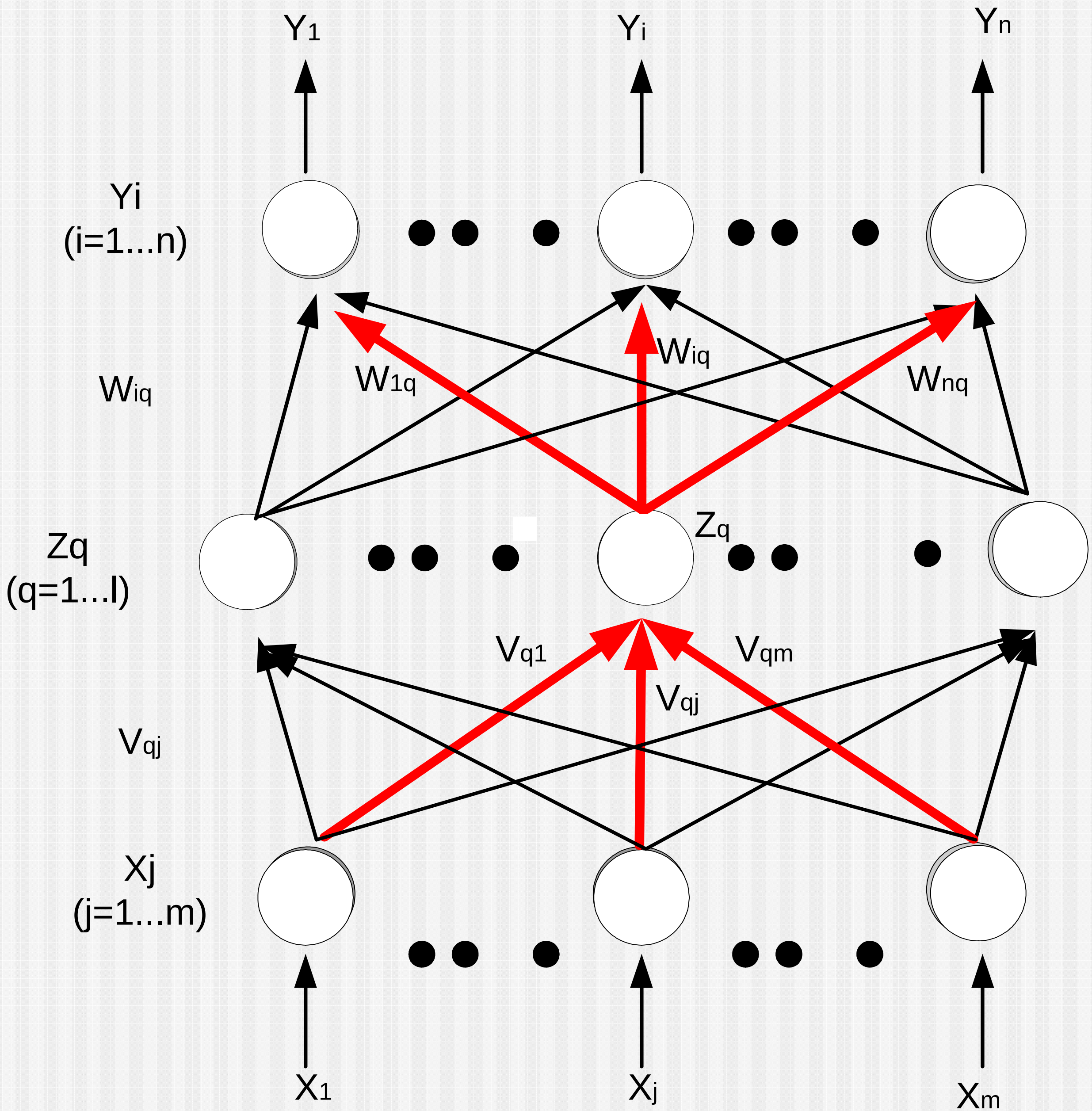

Y1
Yi
Yn
Yi
(i=1...n)
Wiq
W1q
Wiq
Wnq
Zq
(q=1...l)
Zq
Vq1
Vqm
Vqj
Vqj
Xj
(j=1...m)
X1
Xj
Xm

- *ANN (*m *inputs,* n *outputs, 1 hidden layer)*
- *k examples of the form (x,d)*
- *Given an example (x,d),*

$$\text{net}_q = \sum_{j=1}^{m} v_{qj} x_j$$

⇒ *It produces an output*

$$z_q = a(\text{net}_q) = a\left(\sum_{j=1}^{m} v_{qj} x_j\right)$$

⇒ *where a is the activation function.*

- *The net input for a perceptron (PE) i in the output layer is given by*

$$\text{net}_i = \sum_{q=1}^{l} w_{iq} z_q = \sum_{q=1}^{l} w_{iq} a\left(\sum_{j=1}^{m} v_{qj} x_j\right)$$

- *The network produces an output*

$$y_i = a(\text{net}_i) = a\left(\sum_{q=1}^{l} w_{iq} a\left(\sum_{j=1}^{m} v_{qj} x_j\right)\right)$$

Let the cumulative error between the desired and the actual outputs be given by

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(d_i - y_i)^2 = \frac{1}{2}\sum_{i=1}^{n}\left(d_i - a\left(\sum_{q=1}^{l} w_{iq} z_q\right)\right)^2$$

where *E(w)* is a matrix of all the weights.

Using gradient method:

$$\Delta w_{iq} = -\eta \frac{\partial E}{\partial w_{iq}}$$

$$= -\eta \left[\frac{\partial E}{\partial y_i}\right]\left[\frac{\partial y_i}{\partial \mathrm{net}_i}\right]\left[\frac{\partial \mathrm{net}_i}{\partial w_{iq}}\right]$$

$$= \eta\left[d_i - y_i\right]\left[a'(\mathrm{net}_i)\right]\left[z_q\right]$$

$$= \eta \delta_{oi} z_q$$

where δ_{oi} is the error in the i-th output neuron.

$$\delta_{oi} = \frac{\partial E}{\partial \mathrm{net}_i} = \left[d_i - y_i\right]\left[a'(\mathrm{net}_i)\right]$$

$$a'(\mathrm{net}_i) = \frac{\partial a(\mathrm{net}_i)}{\partial \mathrm{net}_i}$$

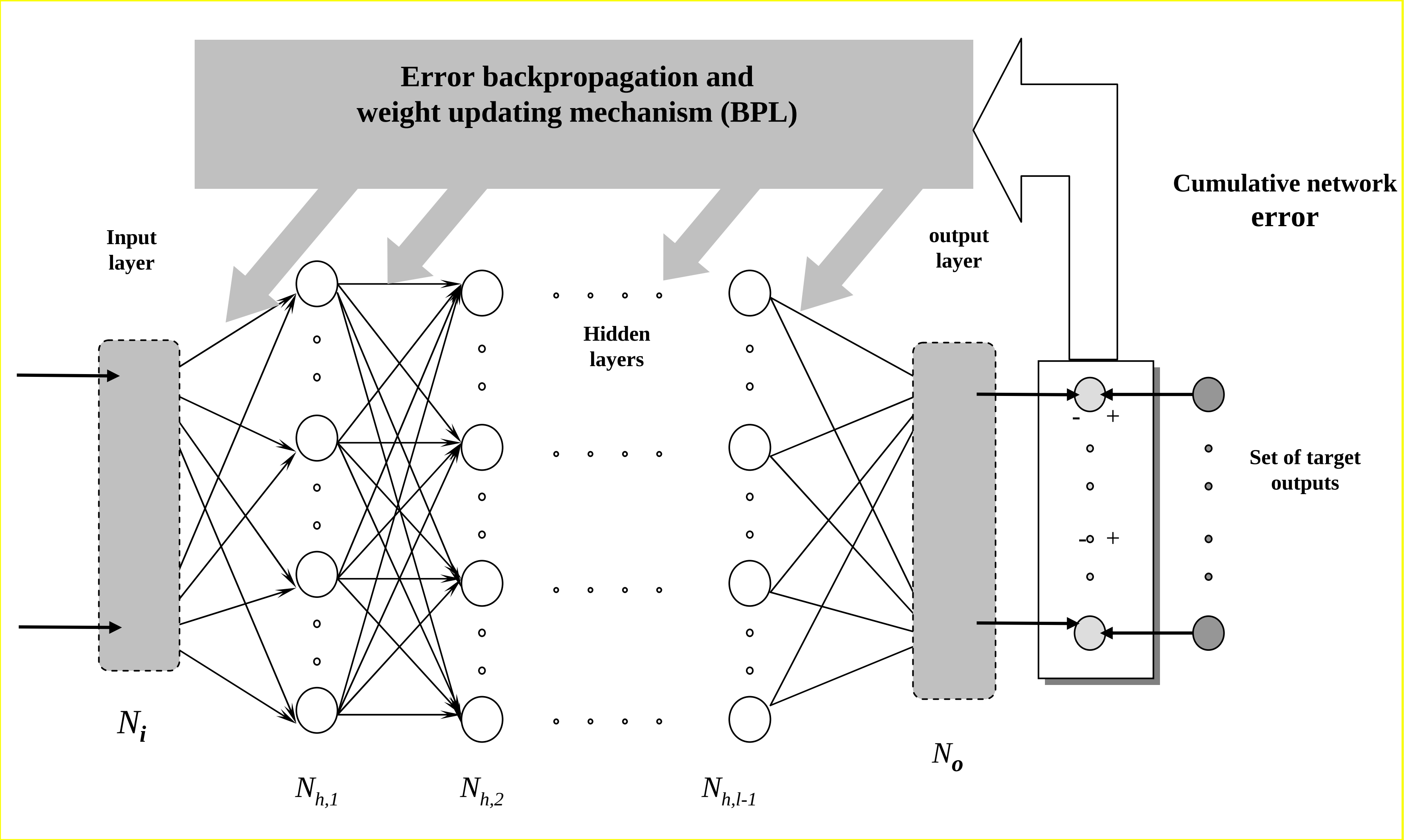
$$\Delta v_{qj} = -\eta\left[\frac{\partial E}{\partial v_{qj}}\right] = \eta \sum_{i=1}^{n}\left[(d_i - y_i)a'(\mathrm{net}_i)w_{iq}\right]a'(\mathrm{net}_q)x_j$$

$$= \eta\delta_{hq}x_j$$

where δ_{hq} is the error in the q-th neuron of the h-th hidden layer.

$$\delta_{hq} = \frac{-\partial E}{\partial \mathrm{net}_q} = a'(\mathrm{net}_q)\sum_{i=1}^{n}\delta_{oi}w_{iq}$$

Backpropagation Algorithm



Consider a ANN with Q *feedforward layers* q=1,…,Q.

${}^{q}net_i$ and ${}^{q}y_i$ *denote the net input and output of the i-th unit in the q-th layer, respectively.*

Let ${}^{q}w_{ij}$ *denote the connection weight from* ${}^{q-1}y_j$ *and* ${}^{q}y_i$*.*

<u>*Input*</u>*: A set of training pairs of* $\left\{(x^{(k)}, d^{(k)}), k = 1, \ldots, p\right\}$

The input vector will be augmented with a last element

$$x_{m+1}^{(k)} = -1$$

<u>*Step 0*</u>*:* (Initialization) *Choose* $\eta > 0$ *and a tolerance error* E_{max}*. Initialize the weights to small random values in [0,1].*

Set E=0 *and* k=1*.*

<u>Step 1</u>: Apply the k-th *exemplar* (q=1)

$$ {}^{q}y_i = {}^{1}y_i = {}^{(k)}x_i $$ *for all* i's

<u>Step 2</u>: (Forward propagation) *Propagate the signal forward*

$$ {}^{q}y_i = a({}^{q}\mathrm{net}_i) = a\left(\sum_j {}^{q}w_{ij}\, {}^{q-1}y_j\right) \quad \text{for } q = 1, \ldots, Q $$

<u>Step 3</u>: (Output error measure) *Compute the error value and the error signals* ${}^{Q}\delta_i$ *for the output layer*

$$ E = \tfrac{1}{2}\sum_{i=1}^{n} (d_i^{(k)} - {}^{Q}y_i)^2 + E $$

$$ {}^{Q}\delta_i = (d_i^{(k)} - {}^{Q}y_i)\, a'({}^{Q}\mathrm{net}_i) $$

Step 4: (Error backpropagation) *Propagate the error signals backward*

$$\Delta\, {}^{q}w_{ij} = \eta\, {}^{q}\delta_i\, {}^{q-1}y_j \qquad {}^{q}w_{ij}^{\text{new}} = {}^{q}w_{ij}^{\text{old}} + \Delta\, {}^{q}w_{ij}$$

$${}^{q-1}\delta_i = a'({}^{q-1}\text{net}_i)\sum_j {}^{q}w_{ji}\, {}^{q}\delta_j \qquad \text{for } q = Q, Q-1, \ldots, 2$$

Step 5: (One epoch looping) *Check whether all the training data has been cycled once.*

If k<p *then* k:= k+1 *and go back to step 1;*

Otherwise go to step 6.

Step 6: (Total error checking)

If $E<E_{max}$ *then process terminated;*

Otherwise ($E \geq E_{max}$*), set* E=0 *and* k=1 *and go back to step 1.*

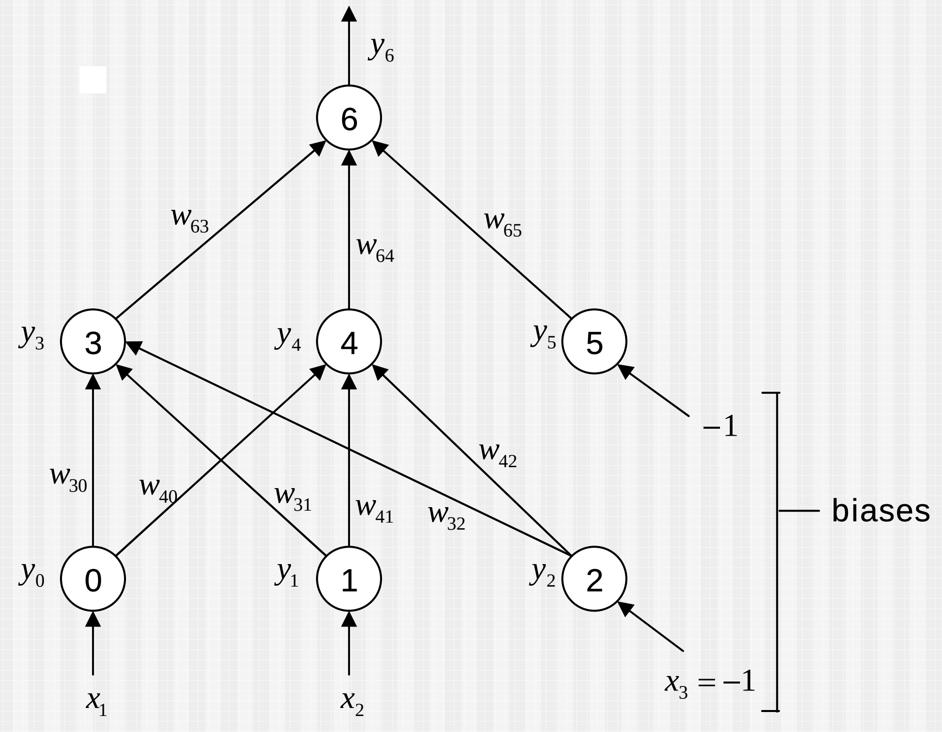
Example

- It is required to train the following network using the back-propagation learning algorithm.
- Each neuron is using a unipolar sigmoid activation function

$$y = a(\text{net}) = \frac{1}{1 + e^{-\lambda \text{net}}}$$

using $\lambda = 1$, then

$$a'(\text{net}) = y(1 - y)$$



Initialization

- Initialize the weights to small random values: Assume all weights are initialized to 0.2
- Set $\eta = 0.2$ (learning rate)
- Set $E_{\max} = 0.01$ (maximum tolerable error)
- Set $E = 0$ (current Error value)
- Set $k = 1$ (current training pattern)

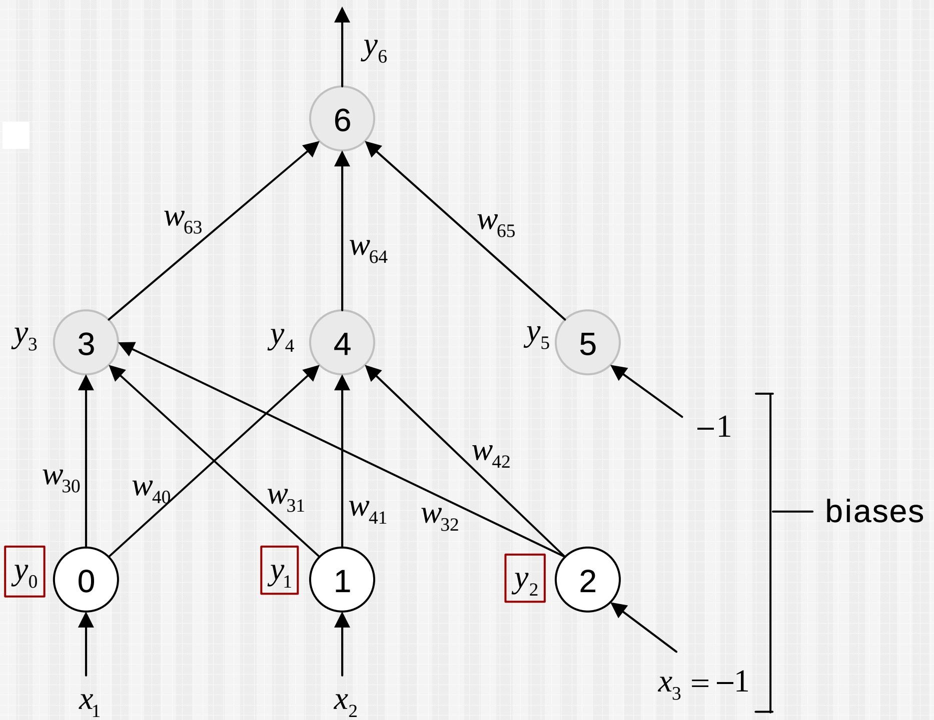
Training loop

Step (1) – Apply Input Pattern

- Apply the *1st* input pattern to the input layer
- Assume

$$\mathbf{x} = (0.3, 0.4) \quad \mathbf{d} = (0.88)$$

$$y_0 = x_1 = 0.3$$
$$y_1 = x_2 = 0.4$$
$$y_2 = x_3 = -1$$

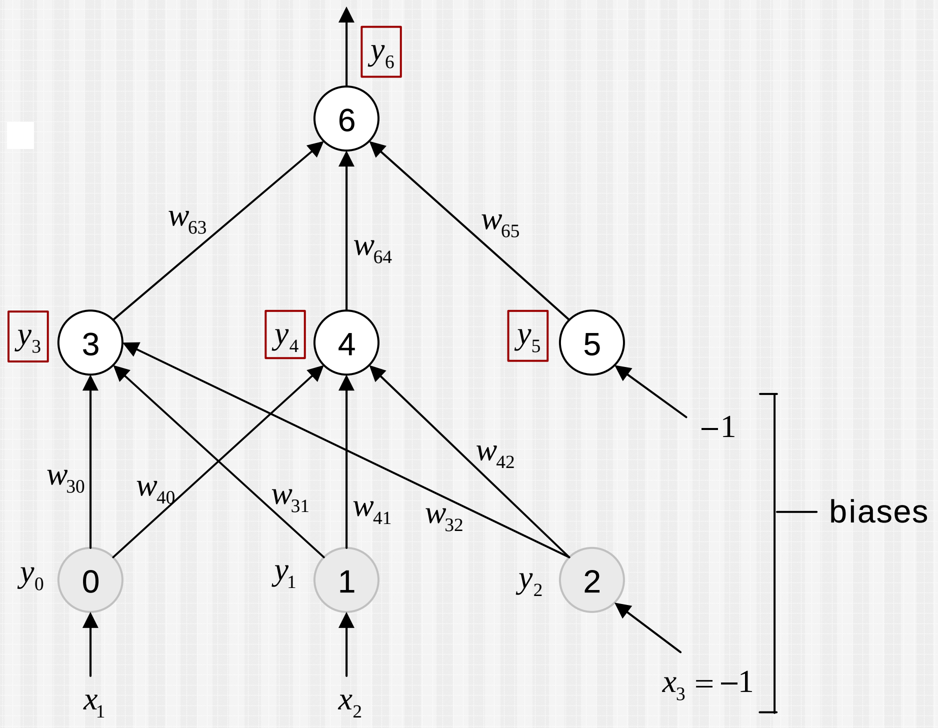


Training loop

Step (2) – Forward Propagation

- Propagate the signal forward through the network

$$
\begin{aligned}
y_3 &= a(w_{30}y_0 + w_{31}y_1 + w_{32}y_2) \\
&= a(0.2(0.3) + 0.2(0.4) + 0.2(-1)) \\
&= a(-0.06) \\
&= 0.485 \\
y_4 &= a(w_{40}y_0 + w_{41}y_1 + w_{42}y_2) \\
&= a(0.2(0.3) + 0.2(0.4) + 0.2(-1)) \\
&= a(-0.06) \\
&= 0.485 \\
y_5 &= -1 \\
y_6 &= a(w_{63}y_3 + w_{64}y_4 + w_{65}y_5) \\
&= a(0.2(0.485) + 0.2(0.485) + 0.2(-1)) \\
&= a(-0.006) \\
&= 0.4985
\end{aligned}
$$

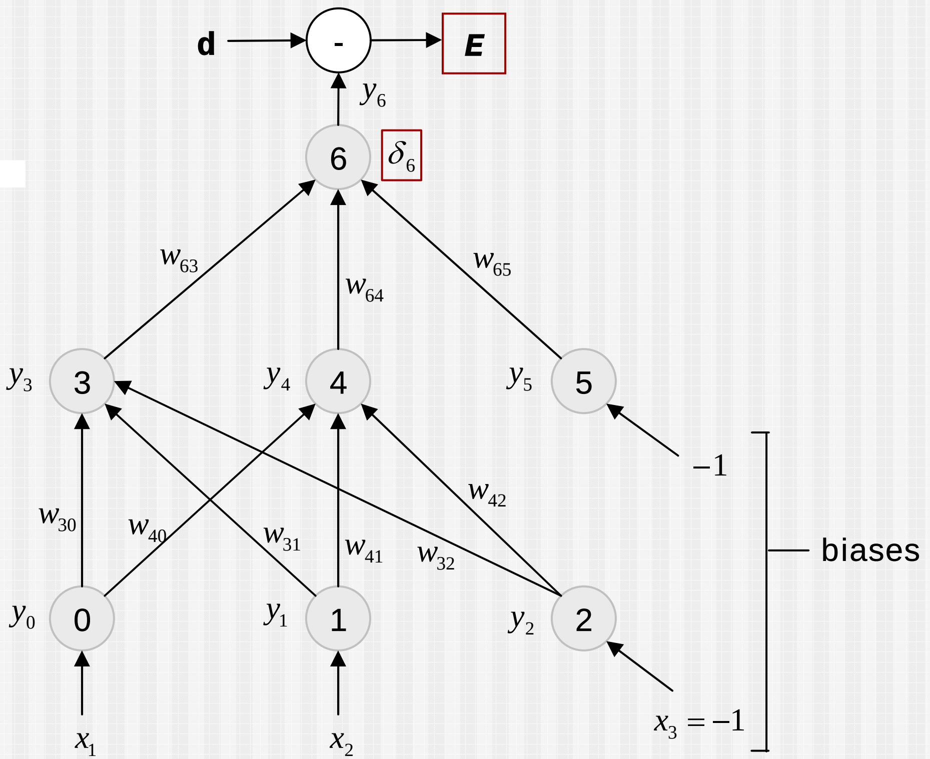


Training loop

Step (3) – Output Error Measure

- Compute the error value and the error signal δ_6 of the output layer

$$
\begin{aligned}
E &= \tfrac{1}{2}(d - y_6)^2 + E \\
&= \tfrac{1}{2}(0.88 - 0.4985)^2 + 0 \\
&= 0.0728 \\
\delta_6 &= a'(\text{net}_6)(d - y_6) \\
&= y_6(1 - y_6)(d - y_6) \\
&= 0.4985(1 - 0.4985)(0.88 - 0.4985) \\
&= 0.0954
\end{aligned}
$$

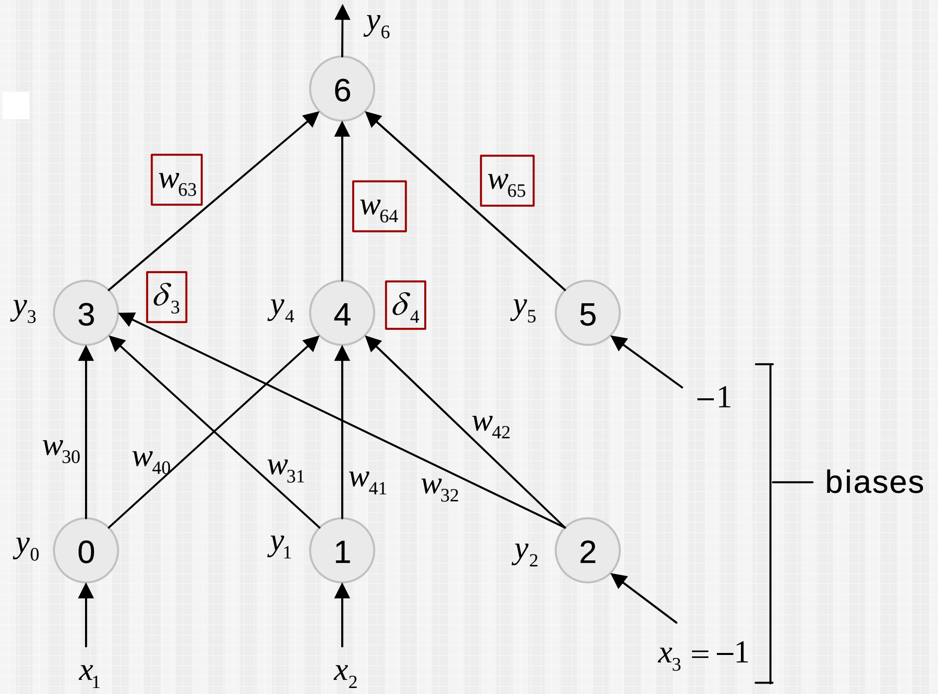


Training loop

Step (4) – Error Back-propagation

- Propagate the errors backward to update the weights and compute the error signals of the preceding layers

$$\Delta w_{63} = \eta \delta_6 y_3 = 0.2(0.0954)(0.485) = 0.0093$$
$$w_{63}^{new} = w_{63}^{old} + \Delta w_{63} = 0.2 + 0.0093 = 0.2093$$
$$\Delta w_{64} = \eta \delta_6 y_4 = 0.2(0.0954)(0.485) = 0.0093$$
$$w_{64}^{new} = w_{64}^{old} + \Delta w_{64} = 0.2 + 0.0093 = 0.2093$$
$$\Delta w_{65} = \eta \delta_6 y_5 = 0.2(0.0954)(-1) = -0.0191$$
$$w_{65}^{new} = w_{65}^{old} + \Delta w_{65} = 0.2 - 0.0191 = 0.1809$$
$$\delta_3 = a'_3(net_3)\sum_{i=6}^{6} w_{i3}\delta_i = y_3(1-y_3)w_{63}\delta_6$$
$$= 0.485(1-0.485)(0.2)(0.0954) = 0.0048$$
$$\delta_4 = a'_4(net_4)\sum_{i=6}^{6} w_{i4}\delta_i = y_4(1-y_4)w_{64}\delta_6$$
$$= 0.485(1-0.485)(0.2)(0.0954) = 0.0048$$



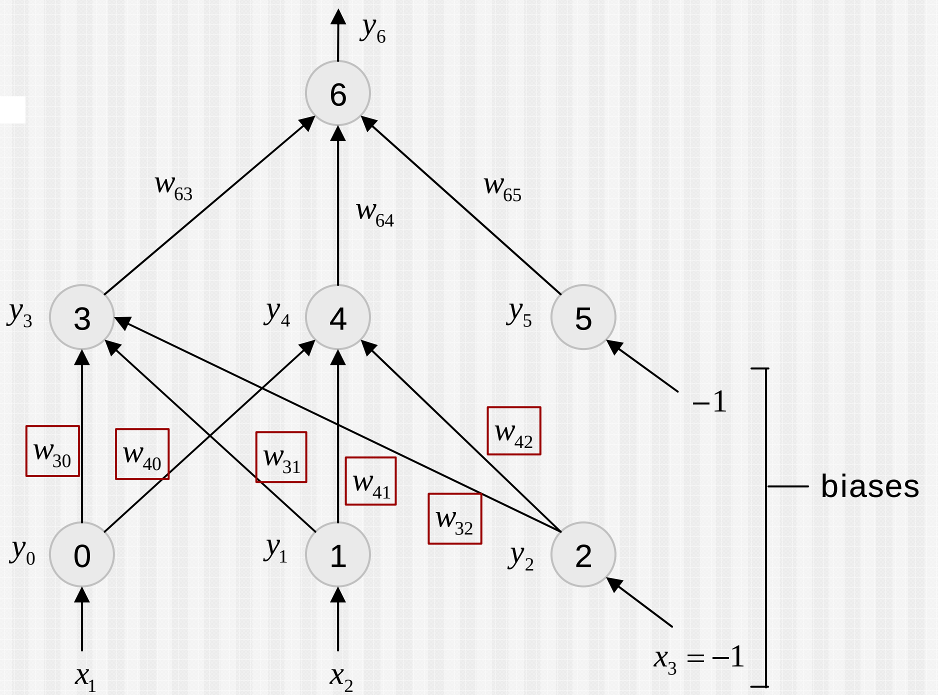
Training loop

Step (4) – Error Back-propagation (cont'd)

- Propagate the errors backward to update the weights and compute the error signals of the preceding layers

$$\Delta w_{30} = \eta\delta_3 y_0 = 0.2(0.0048)(0.3) = 0.000288$$
$$w_{30}^{\text{new}} = w_{30}^{\text{old}} + \Delta w_{30} = 0.2 + 0.000288 = 0.200288$$
$$\Delta w_{31} = \eta\delta_3 y_1 = 0.2(0.0048)(0.4) = 0.000384$$
$$w_{31}^{\text{new}} = w_{31}^{\text{old}} + \Delta w_{31} = 0.2 + 0.000384 = 0.200384$$
$$\Delta w_{32} = \eta\delta_3 y_2 = 0.2(0.0048)(-1) = -0.00096$$
$$w_{32}^{\text{new}} = w_{32}^{\text{old}} + \Delta w_{32} = 0.2 - 0.00096 = 0.19904$$

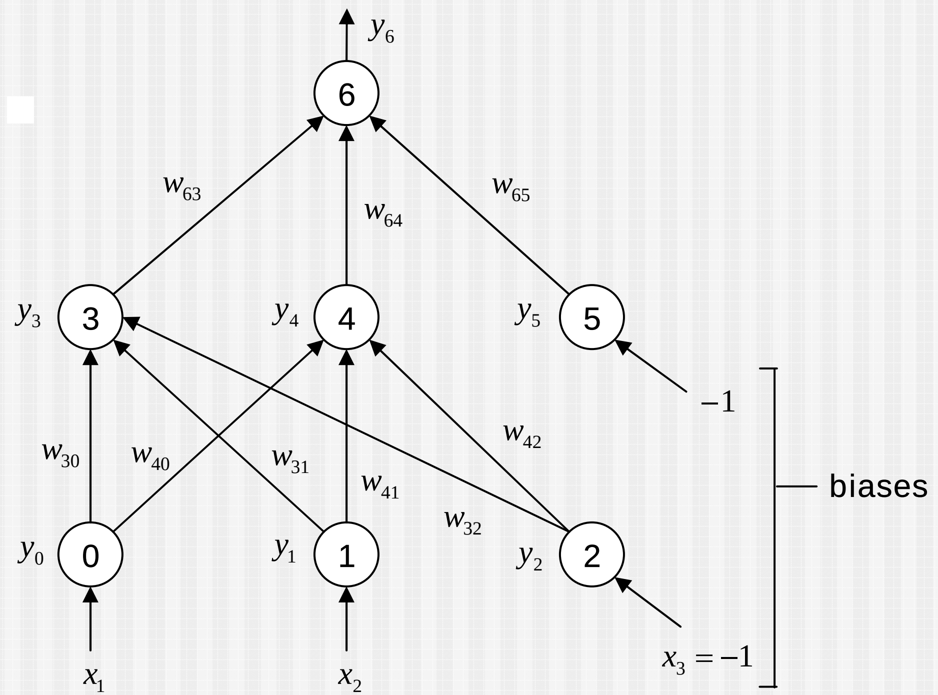
$$\Delta w_{40} = \eta\delta_4 y_0 = 0.2(0.0048)(0.3) = 0.000288$$
$$w_{40}^{\text{new}} = w_{40}^{\text{old}} + \Delta w_{40} = 0.2 + 0.000288 = 0.200288$$
$$\Delta w_{41} = \eta\delta_4 y_1 = 0.2(0.0048)(0.4) = 0.000384$$
$$w_{41}^{\text{new}} = w_{41}^{\text{old}} + \Delta w_{41} = 0.2 + 0.000384 = 0.200384$$
$$\Delta w_{42} = \eta\delta_4 y_2 = 0.2(0.0048)(-1) = -0.00096$$
$$w_{42}^{\text{new}} = w_{42}^{\text{old}} + \Delta w_{42} = 0.2 - 0.00096 = 0.19904$$



Training loop

Step (5) – One Epoch Looping

- Check whether the whole set of training data has been cycled once. If $k<p$ then $k=k+1$ and go to Step (1); otherwise go to Step (6).
- For this example we should continue with the next training pattern and loop again from Step (1)



Training loop

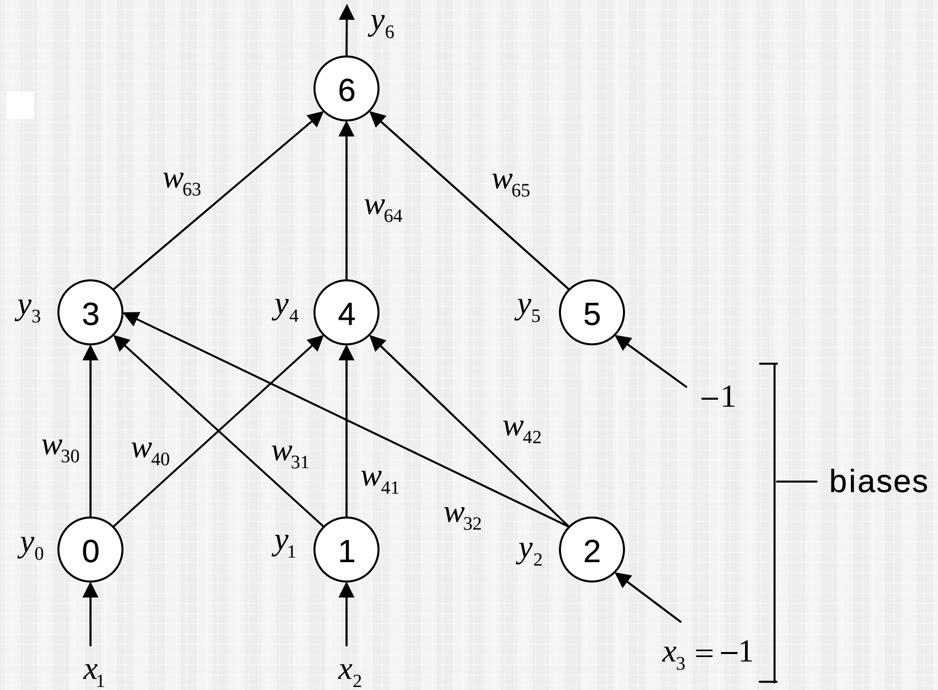
Step (6) – Total Error Checking

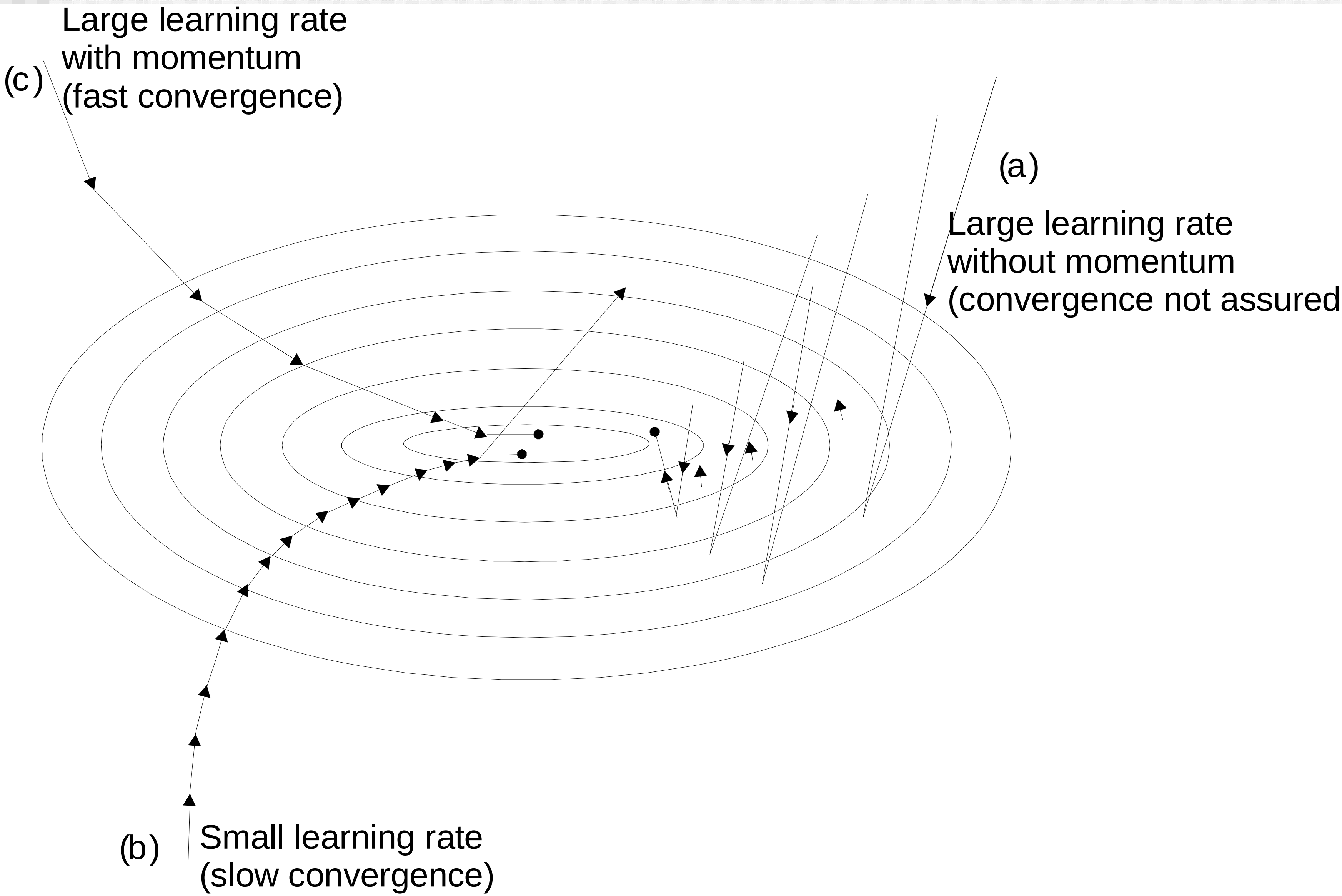
- Check whether the current total error is acceptable. If $E < E_{\max}$ then terminate the training process and output the final weights; otherwise *E=0, k=1,* and initiate the new training Epoch by going to Step (1).
- In this example:

$$E = 0.61,\ E_{\max} = 0.01$$

$$E \text{ is not less than } E_{\max}$$

then we continue with the next epoch by cycling the training data again.




(c)
Large learning rate
with momentum
(fast convergence)
(a)
Large learning rate
without momentum
(convergence not assured)
(b)
Small learning rate
(slow convergence)