EENG582: Artificial Neural Networks

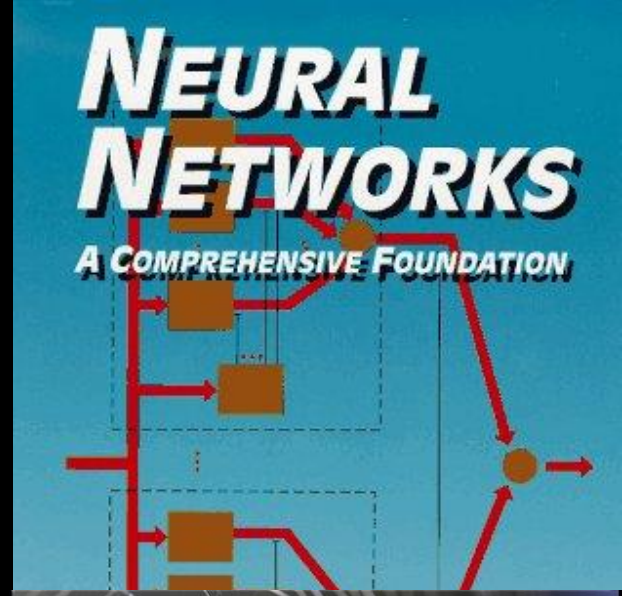
Neural Networks
A Comprehensive Foundation
by S. Haykin

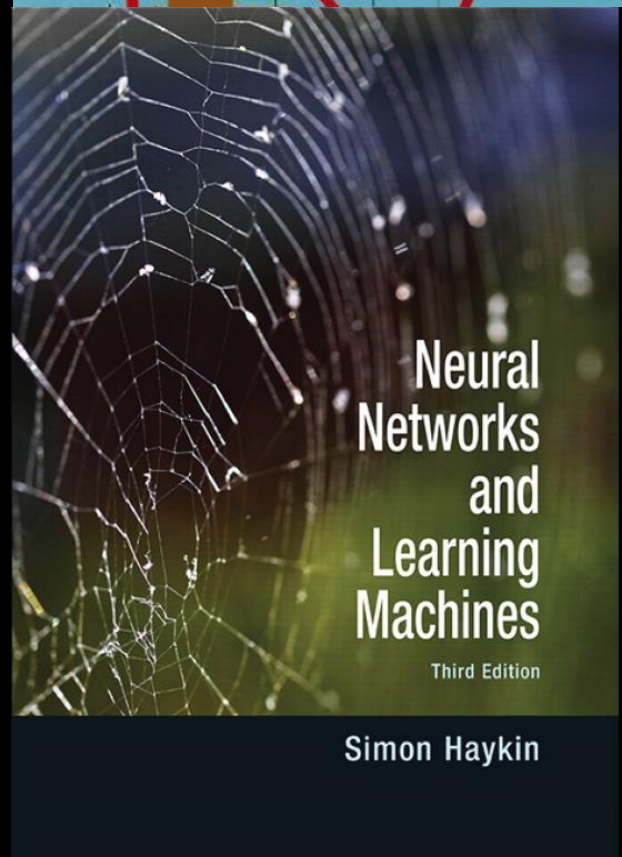
Sections 4.1 – 4.4

4 Multilayer Perceptrons

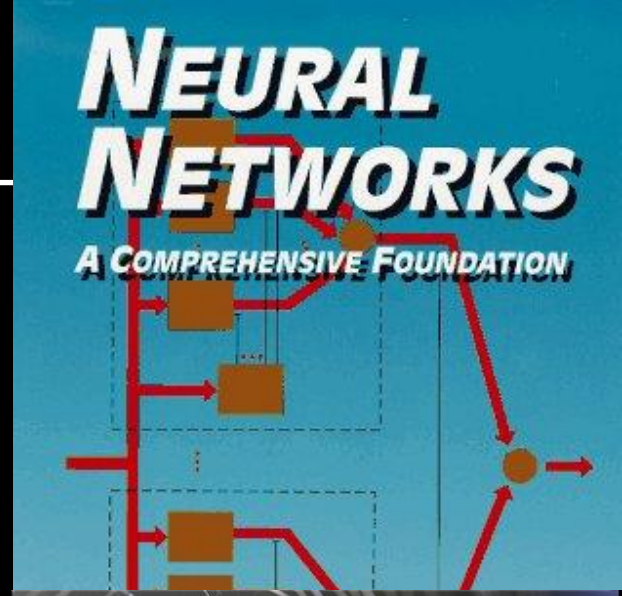
Prof. Dr. Hasan AMCA
Electrical and Electronic Engineering Department
(ee.emu.edu.tr)

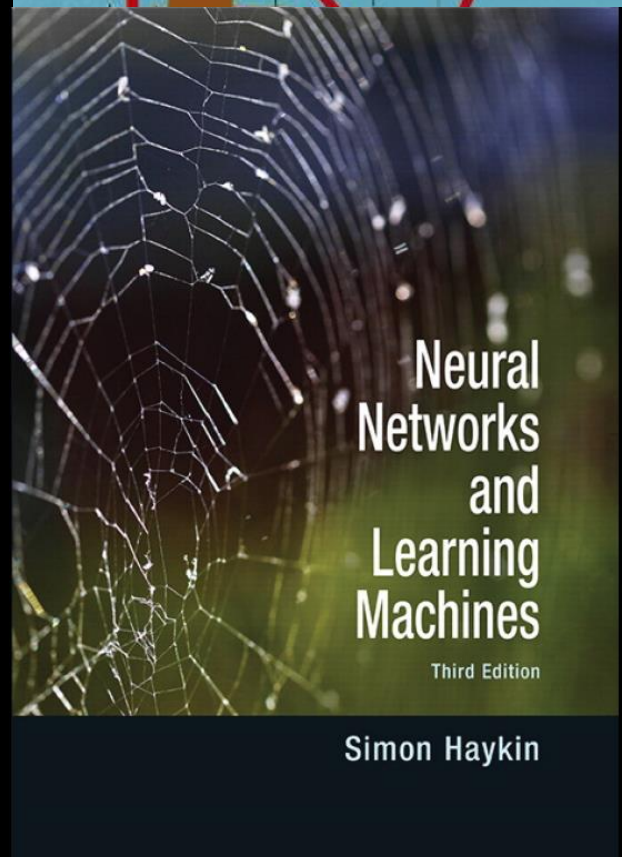
Eastern Mediterranean University
(emu.edu.tr)





4 Multi-Layer Perceptrons





4.1 Introduction

- The class of neural networks
  - which consists of a set of source nodes called the input layer,
  - one or more hidden layers of computation nodes and
  - an output layer of computation nodes.
- These neural networks, where the signal propagates in forward direction, are commonly referred to as *multilayer perceptrons* (MLPs), which represent a generalization of the single-layer perceptron.
- Multilayer perceptrons have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner with *the error back-propagation algorithm*.
- This algorithm is based on the error-correction learning rule which may be viewed as a generalization of adaptive filtering algorithm:
  - the least-mean-square (LMS) algorithm was described earlier for the special case of a single linear neuron.

4.1 Introduction

- The error back-propagation learning consists of two passes through the different layers of the network:

  - a forward pass and a backward pass.

- In the forward pass, an activity pattern (input vector) is applied to the sensory nodes of the network, and its effect propagates through the network layer by layer.
- Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the networks are all fixed.
- During the backward pass, the synaptic weights are all adjusted in accordance with an error-correction rule.

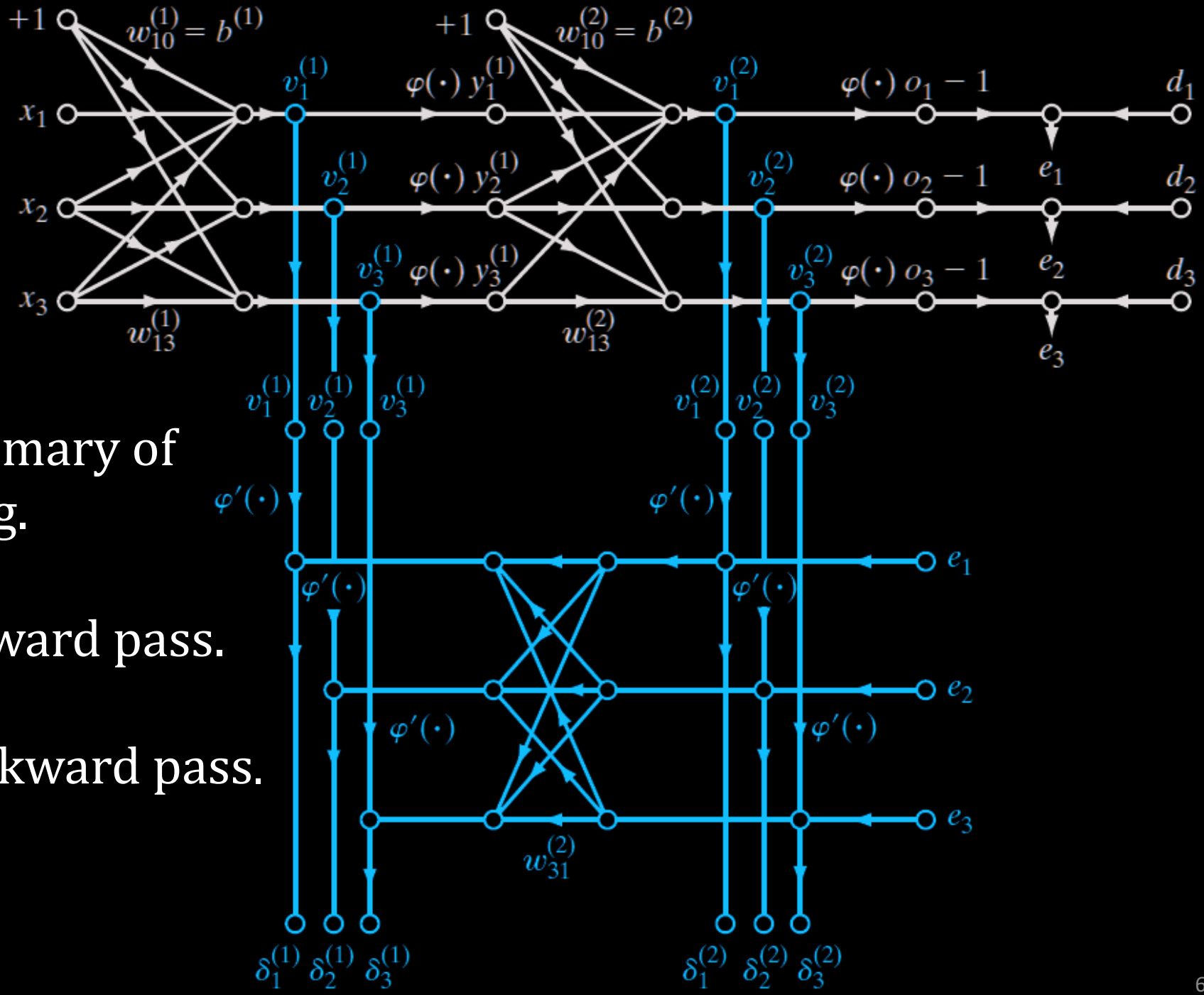
4.1 Introduction

- Specifically, the actual response of the network is subtracted from a desired (target) response to produce an error signal.
- This error signal is then propagated backward through the network against direction of synaptic connections hence the name "error back-propagation".
  - Or the back-propagation algorithm, or simply backprop.
  - the related learning process is back-propagation learning.
- The synaptic weights are adjusted to make the actual response of the network move closer to the desired response in a statistical sense.
- The back propagation process is shown in Fig. 4.1, which will be revisited later.

Signal-flow graphical summary of back-propagation learning.

Top part of the graph: forward pass.

Bottom part of graph: backward pass.



4.1 Introduction

- A multilayer perceptron has three distinctive characteristics:

1. The model of each neuron in the network includes a nonlinear (smooth, differentiable) activation function such as a sigmoidal nonlinearity defined by :

$$y_j = \frac{1}{1 + \exp(-v_j)}$$

where v_j is the induced local field. Without nonlinearity, the input-output relation of the network reduces to that of a single-layer perceptrons.

2. The network contains one or more layers of *hidden neurons* which enable the network to learn complex tasks by extracting progressively more meaningful features from the input patterns (vectors).

3. The network exhibits a high degrees of *connectivity*, determined by the synapses of the network. A change in the connectivity of the network requires a change in the population of synaptic connections or their weights.

4.2 Architecture of a Multilayer Perceptron

- Fig. 4.1 shows the architectural graph of a multilayer perceptron with two hidden layers and an output layer forming a fully connected network.
- Signal flows through the network in a forward direction, from left to right and on a layer-by-layer basis.

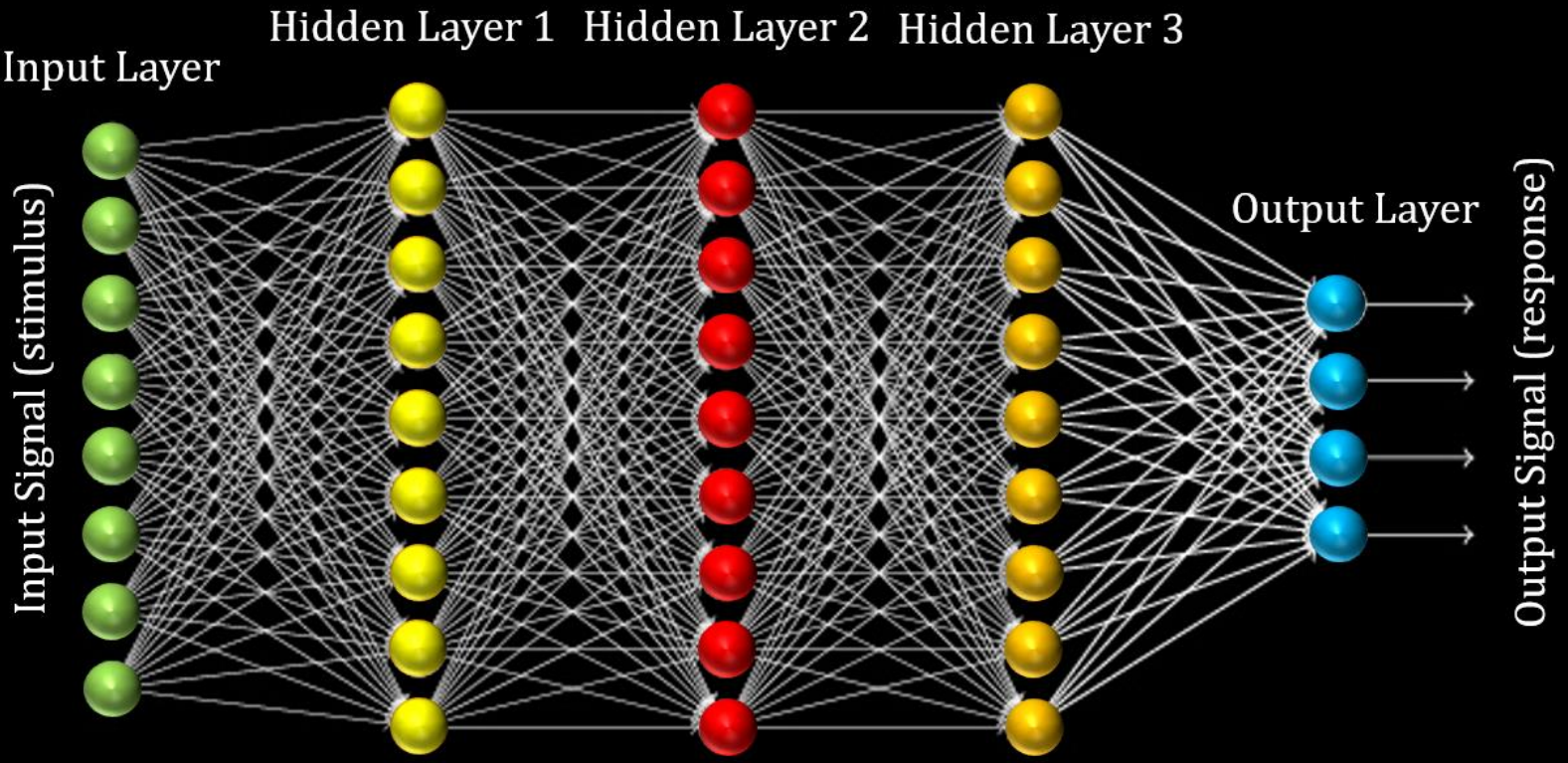


Fig. 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

4.2 Architecture of a Multilayer Perceptron

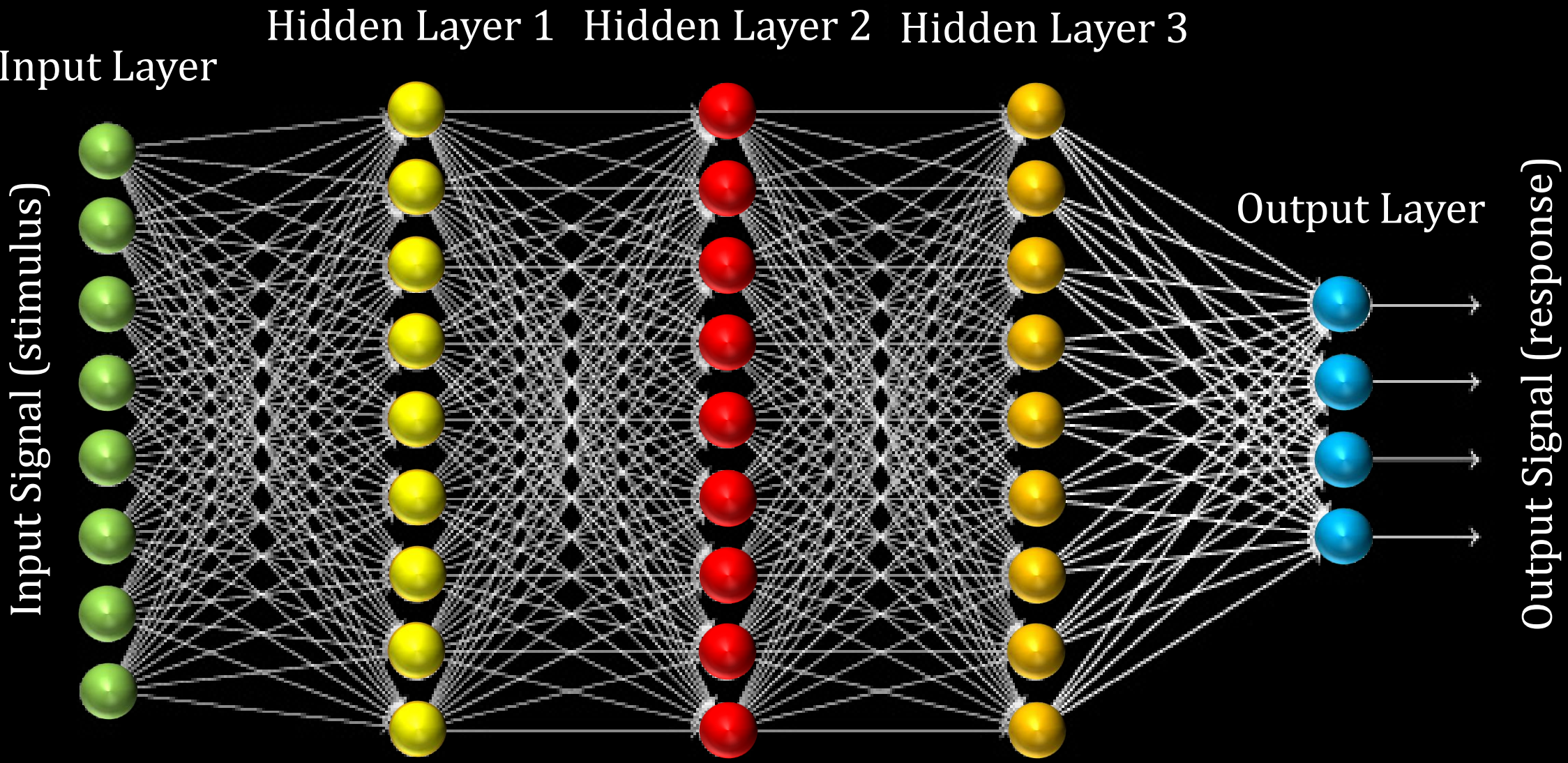
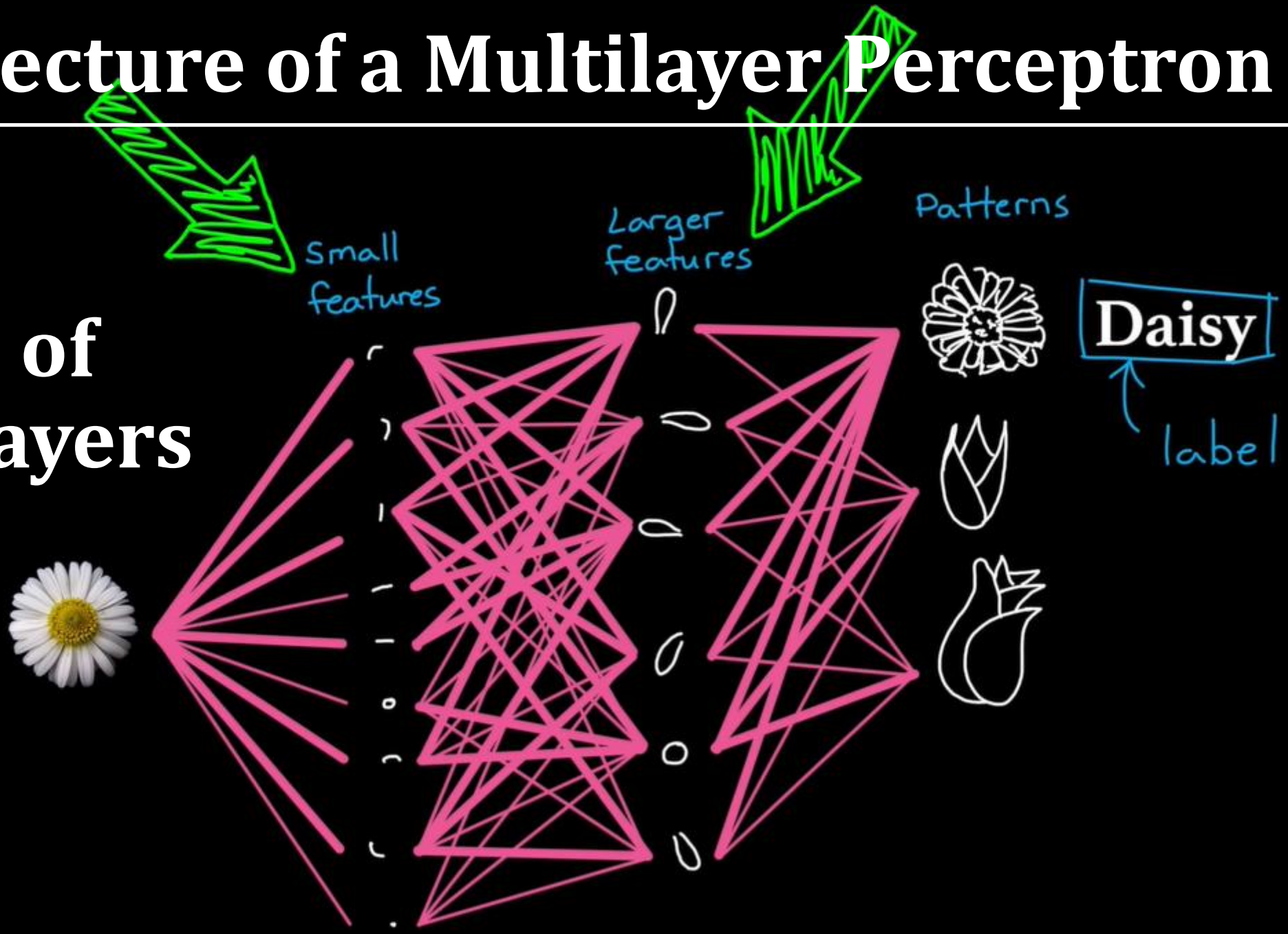


Fig. 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

4.2 Architecture of a Multilayer Perceptron

The Use of Multiple Layers



https://www.mathworks.com/videos/deep-learning-for-engineers-part-1-why-choose-deep-learning-1617287683265.html

4.2 Some Preliminaries

- Fig. 4.2 depicts a portion of the multilayer perceptron. Two kinds of signals are identified in this network.
  1. *Function Signals*. A function signal is an input signal (stimulus) that is input to the network, propagates forward (neuron by neuron) through the network, and emerges at the output end of network as an output signal.
  2. *Error Signals*. An error signal originates at an output neuron of the network and propagates backward (layer by layer) through the network.
- The output neurons (computational nodes) constitute the output layers of the network.
- Remaining neurons (computational nodes) constitute hidden layers.

4.2 Some Preliminaries

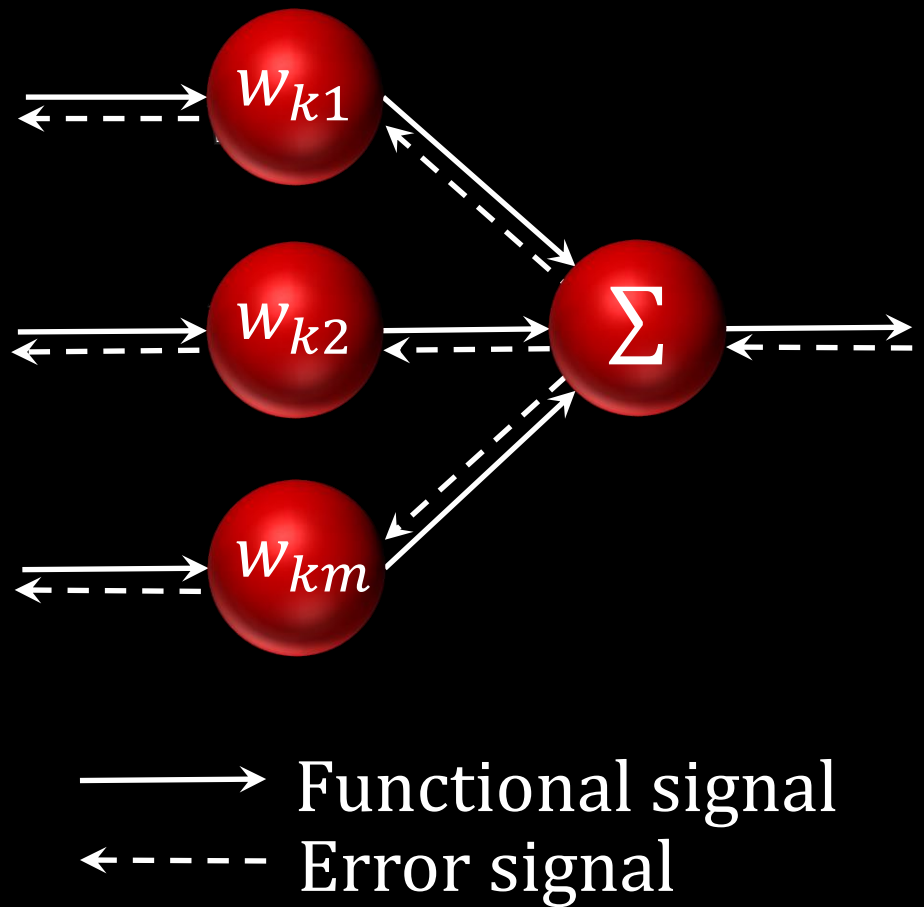


Fig. 4.2 Illustration of the directions of two basic signal flows in a multilayer perceptron: forward-propagation of function signals and back-propagation of error signals.

4.2 Some Preliminaries

- Each hidden or output neuron of a multilayer perceptron performs two computations:
  1. The computation of the function signal appearing at the output of a neuron, which is expressed as a continuous nonlinear function of the input signal and synaptic weights associated with that neuron.
  2. The computation of an estimate of the gradient vector (i.e., the gradients of the error surface with respect to the weights connected to the inputs of a neuron), which is needed for the backward pass through the network.

Notations Used in Derivation of Back-propagation Algorithm

- The indices i, j and k refer to different signals propagating through the network from left to right.
- In iteration (time step) n, the nth training pattern (example) is presented to the network.
- The symbol $\mathcal{C}(n)$ refers to the instantaneous sum of error squares or error energy at iteration n. The average of $\mathcal{C}(n)$ over all values of n (i.e., the entire training set) yields the average error energy $\mathcal{C}_{av}$.
- The symbol $e_j(n)$ refers to the error signal at the output of neuron j for iteration n.
- The symbol $d_j(n)$ refers to the desired response for neuron j and is used to compute $e_j(n)$.

Notations Used in Derivation of Back-propagation Algorithm

- The symbol $y_j(n)$ refers to the function signal appearing at the output of neuron j at iteration n.
- The symbol $w_{ji}(n)$ denotes the synaptic weight connecting the output of neuron i to the input of neuron j at iteration n. The correction applied to this weight at iteration n is denoted by $\Delta w_{ji}(n)$.
- The induced local field of neuron j at iteration n is denoted by $v_j(n)$; it constitutes the signal applied to the activation function associated with neuron j.
- The activation function describing the input-output functional relationship of the nonlinearity associated with neuron j is denoted by $\varphi(.)$.

Notations Used in Derivation of Back-propagation Algorithm

- The bias applied to neuron j is denoted by b_j; its effect is represented by a synapse of weight $w_{j0} = b_j$ connected to a fixed input equal to + 1.
- The ith element of the input vector (pattern) is denoted by $x_i(n)$.
- The kth element of the overall output vector (pattern) is denoted by $o_k(n)$.
- The learning-rate parameter is denoted by η·
- The symbol m_l denotes the size (i.e., number of nodes) in layer l of the multilayer perceptron; l = 0, 1, ... , L, where L is the "depth" of the network.
- Thus m_0 denotes the size of the input layer, m_1 denotes the size of the first hidden layer, and m_L denotes the size of the output layer.
- The notation $m_L = M$ is also used.

4.3 Back-Propagation Algorithm

- The error signal at the output of neuron j at iteration n (i.e., presentation of the nth training example) is defined by

$$e_j(n) = d_j(n) - y_j(n) \text{ neuron } j \text{ is an output node} \qquad (4.1)$$

- The instantaneous value of the total error energy $\mathcal{E}(n)$ is obtained by
- summing the instantaneous value of the error energy for neuron j as $\frac{1}{2}e_j^2(n)$ over all neurons in the output layer;
  - these are the only "visible" neurons for which error signals can be calculated directly.

Supplementary Materials
What is backpropagation really doing? | Deep learning, chapter 3
https://www.youtube.com/watch?v=Ilg3gGewQ5U

4.3 Back-Propagation Algorithm

- We may thus write

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n) \qquad (4.2)$$

- where the set C includes all the neurons in the output layer of the network.
- The average squared error energy is obtained by summing$\mathcal{E}(n)$over all n and then normalizing with respect to the set size N, as shown by

$$\mathcal{E}_{av} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{E}(n) \qquad (4.3)$$

- N denote total number of patterns (examples) contained in training set.

4.3 Back-Propagation Algorithm

- The objective of the learning process is to adjust the free parameters of the network to minimize $\mathcal{E}_{av}$.
- To do this minimization, we consider a simple method of training in which the weights are updated on a pattern-by-pattern basis until one epoch, that is, one complete presentation of the entire training set has been dealt with.
- The adjustments to the weights are made in accordance with the respective errors computed for each pattern presented to the network.
- Consider then Fig. 4.3. which depicts neuron *j* being fed by a set of function signals produced by a layer of neurons to its left.

4.3 Back-Propagation Algorithm

- The induced local field $v(n)$ produced at the input of the activation function associated with neuron j is therefore

$$vJ(n) = \sum_{i=0}^{m} w_{ji}(n) y_j(n) \qquad (4.4)$$

- where m is total number of inputs (excluding the bias) applied to neuron j.
- The synaptic weight w_{j0} (corresponding to the fixed input $y_0 = +1$) equals the bias b_j applied to neuron i.
- Hence the function signal $y_j(n)$ appearing at the output of neuron j at iteration n is

$$y_j(n) = \varphi(v_j(n)) \qquad (4.5)$$

4.3 Back-Propagation Algorithm

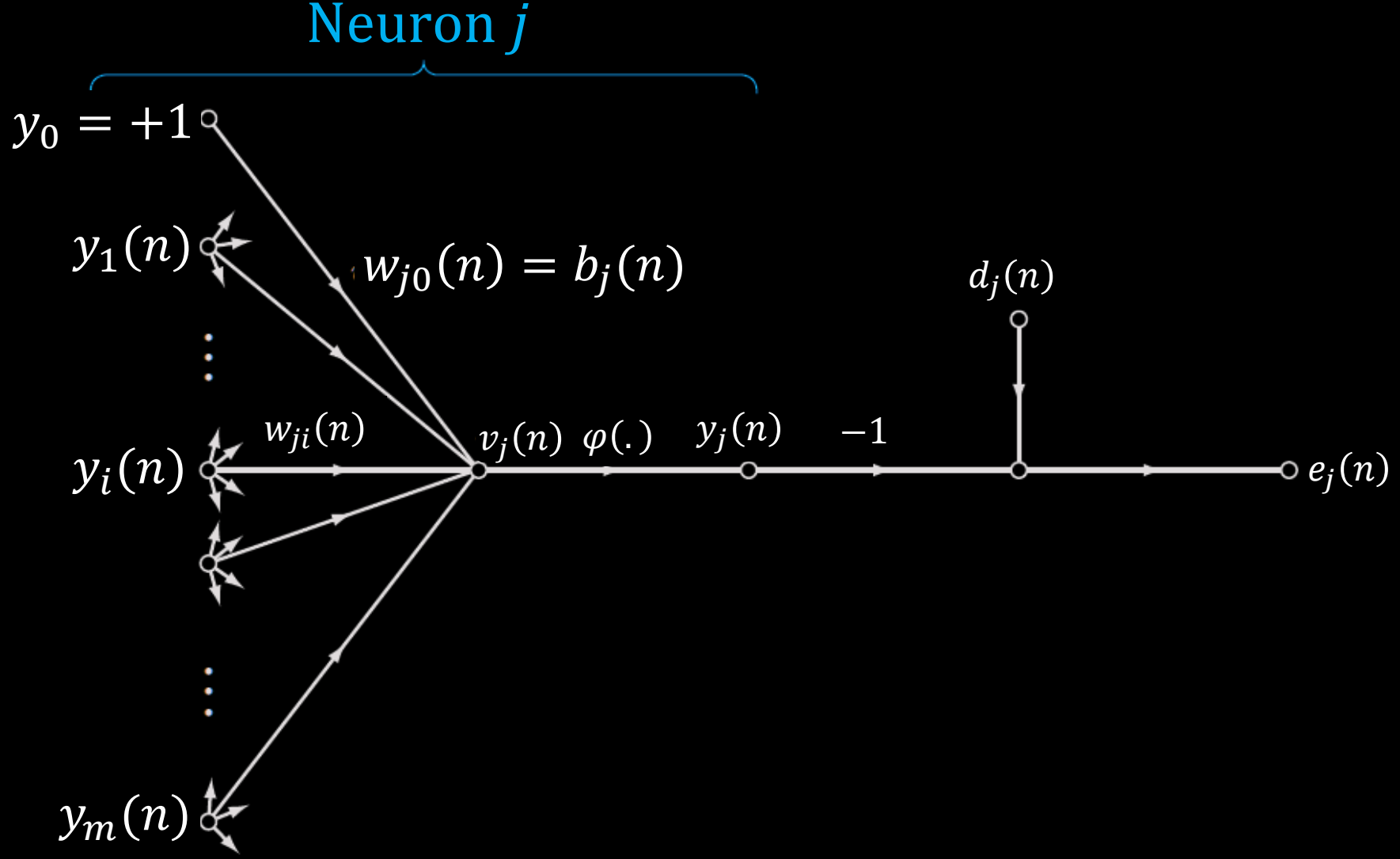


Fig. 4.3 Signal-flow graph highlighting the details of output neuron *j*.

4.3 Back-Propagation Algorithm

- In a manner similar to the LMS algorithm, the back-propagation algorithm applies a correction $\Delta w_{ji}(n)$ to the synaptic weight $w_{ji}(n)$, which is proportional to the partial derivative $\partial \mathcal{E}(n)/\partial w_{ji}(n)$
- This gradient is expressed as:

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{4.46}$$

- Partial derivative $\partial \mathcal{E}(n)/\partial w_{ji}(n)$ represents a *sensitivity factor*, determining the direction of search in weight space for the synaptic weight w_{ji}.

4.3 Back-Propagation Algorithm

- Differentiating both sides of (4.2) with respect to $e_j(n)$, we get

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n) \tag{4.7}$$

- Differentiating both sides of (4.2) with respect to $y_j(n)$, we get

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{4.8}$$

- Differentiating both sides of (4.2) with respect to $\mathrm{v}_j(n)$, we get

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \tag{4.9}$$

- where the prime on r.h.s. signifies differentiation with respect to the argument. Finally, differentiating Eq. (4.4) with respect to $w_{ji}(n)$ yields

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n) \tag{4.10}$$

4.3 Back-Propagation Algorithm

- The use of Eqs. (4.7) to (4.10) in (4.6) yields

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'\Big(v_j(n)\Big)y_i(n) \tag{4.11}$$

- The correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by *delta rule*:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} \tag{4.12}$$

- where η is the learning-rate parameter of the back-propagation algorithm.
- Use of the minus sign in (4.12) accounts for gradient descent in weight space (i.e., seeking a direction for weight change that reduces the value of $\mathcal{E}(n)$.
- Accordingly, the use of (4.11) in (4.12) yields

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_j(n) \tag{4.13}$$

4.3 Back-Propagation Algorithm

- where the local gradient $\delta_j(n)$ is defined by

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = -\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)\, \varphi_j'(v_j(n)) \qquad (4.14)$$

- The local gradient points to required changes in synaptic weights.
- From (4.13) and (4.14) we note that a key factor involved in tile calculation
- of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j.
- In this context we may identify the following two distinct cases, depending on where in the network neuron j is located.

4.3 Back-Propagation Algorithm

Case 1 Neuron *j* is an Output Node

- When neuron *j* is located in the output layer of the network, it is supplied wjth a desired response of its own.
- We may use (4.1) to compute the error signal $e_j(n)$ associated with this neuron.
- We can compute the local gradient $\delta_j(n)$ using (4.14).

4.3 Back-Propagation Algorithm

Case 2 Neuron *j* is a Hidden Node

- When neuron *j* is located in a hidden layer of the network, there is no specified desired response for that neuron.
- The error signal for a hidden neuron would be determined recursively in terms of the error signals of all the neurons to which that hidden neuron is directly connected.
- Consider the situation depicted in Fig. 4.4, which depicts neuron *j* as a hidden node of the network. According to (4.14), we may redefine the local gradient $\delta_j(n)$ for hidden neuron *j* as

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}\varphi_j'(v_j(n)) \text{ neuron } j \text{ is hidden} \qquad (4.15)$$

4.3 Back-Propagation Algorithm

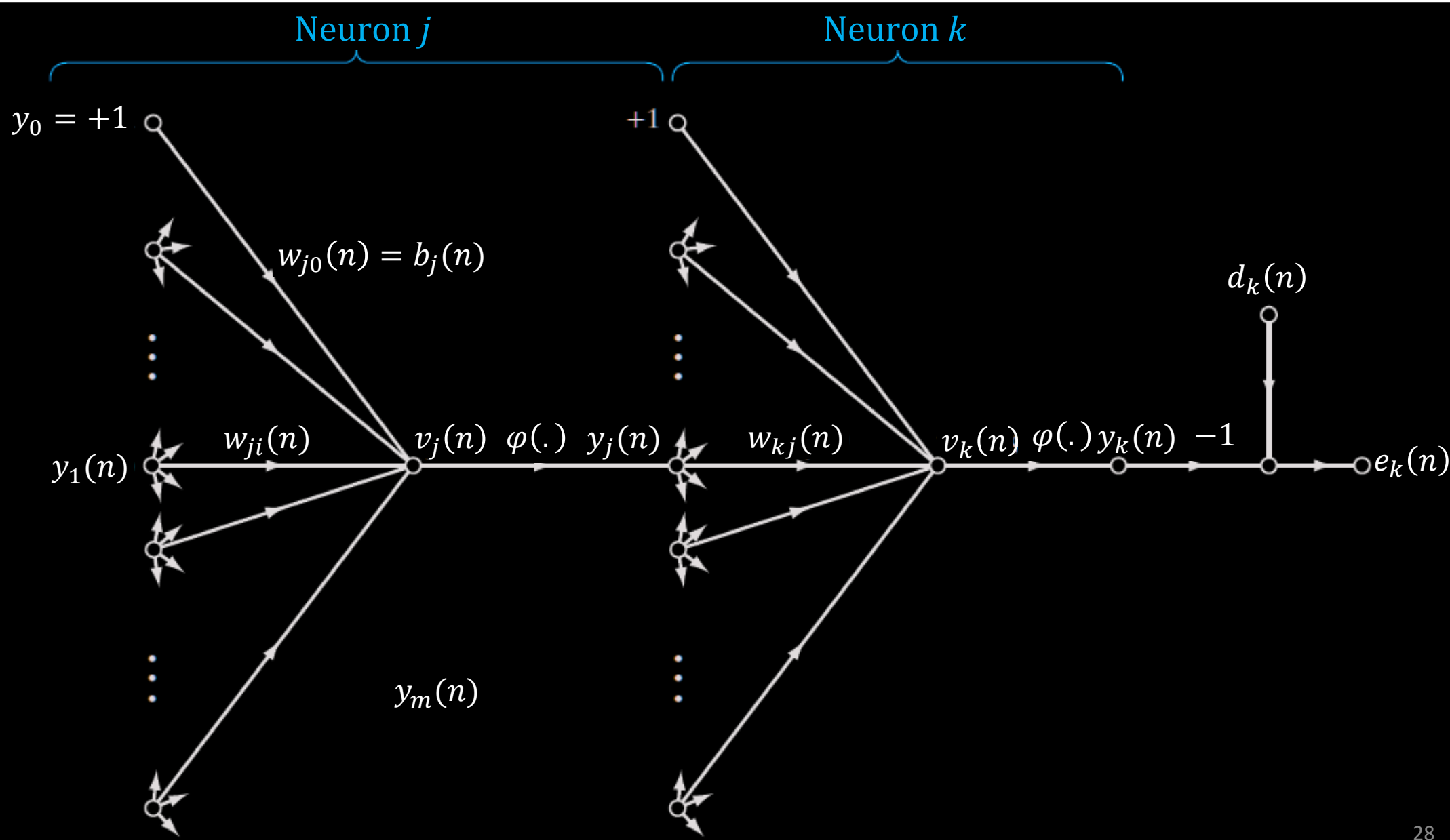


Fig. 4.4 Signal-flow graph highlighting the details of output neuron k connected to hidden neuron j.

4.3 Back-Propagation Algorithm

- After vigorous mathematics, we get the back-propagation formula for the local gradient $\delta_j(n)$ as described:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \quad \text{neuron } j \text{ is hidden} \qquad (4.24)$$

- Figure 4.5 shows the signal-flow graph representation of (4.24), assuming that the output layer consists of $m_{l,}$ neurons.

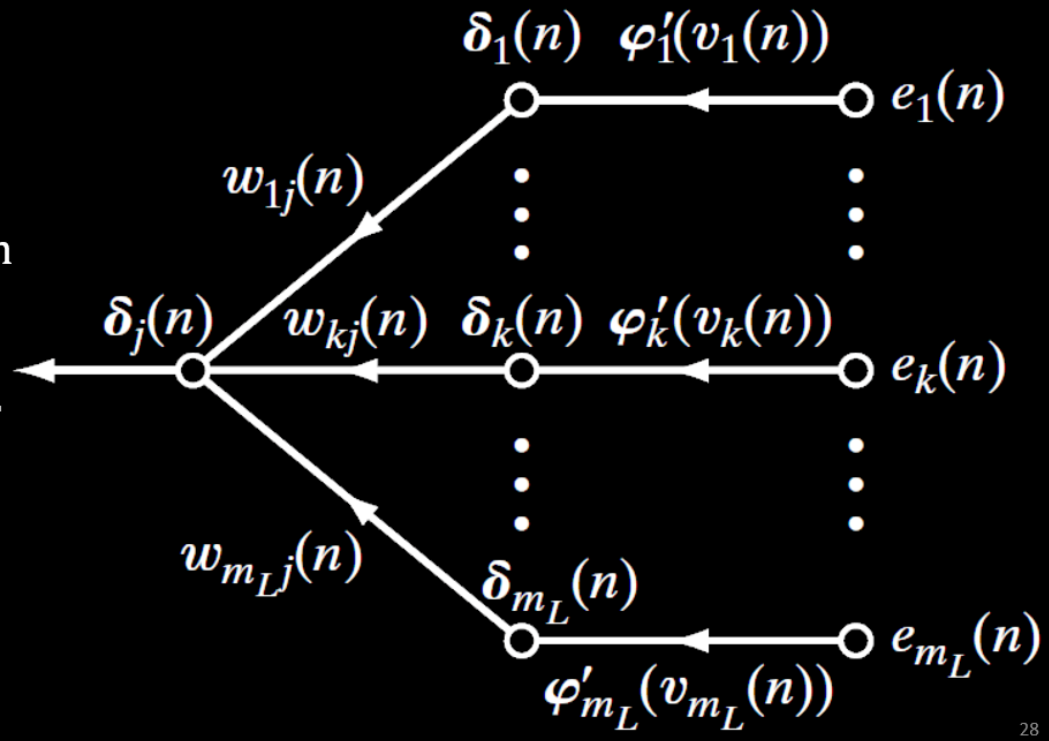


Fig. 4.5 Signal-flow graph of a part of the adjoint system pertaining to backpropagation of error signals.

4.3 Back-Propagation Algorithm

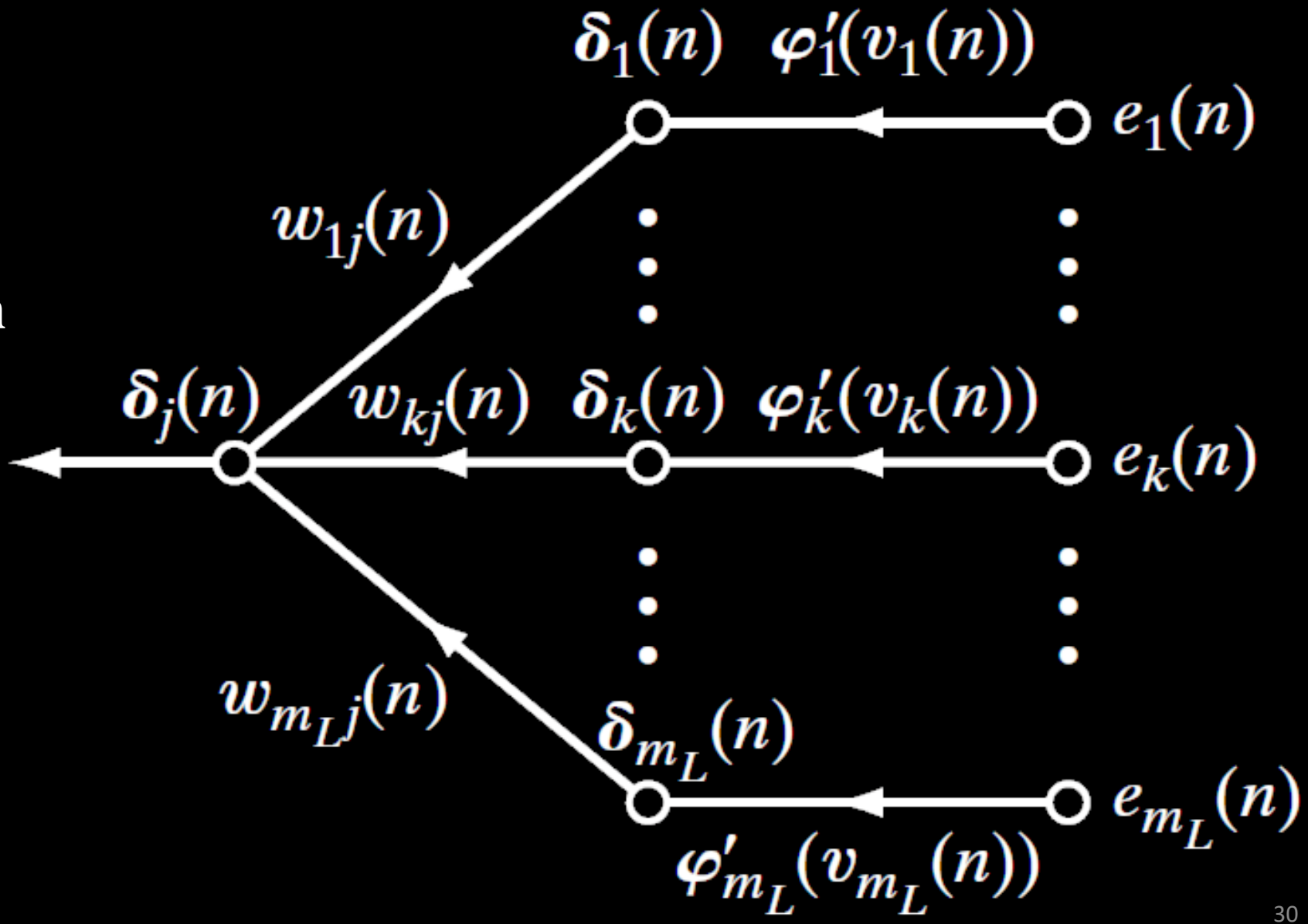


Fig. 4.5 Signal-flow graph of a part of the adjoint system pertaining to backpropagation of error signals.

4.3 Back-Propagation Algorithm

- The factor $\varphi_j'(v_j(n))$ involved in the computation of the local gradient $\delta_j(n)$ in (4.24) depends solely on the activation function associated with hidden neuron *j*.
- The remaining factor involved in this computation, namely the summation over k, depends on two sets of terms.
- The first set of terms, the $\delta_k(n)$, requires knowledge of the error signals $e_k(n)$, for all neurons that lie in the layer to the immediate right of hidden neuron *j*, and that are directly connected to neuron *j*: see Fig. 4.4.
- The second set of terms, the $w_{kj}(n)$, consists of the synaptic weights associated with these connections.

4.3 Back-Propagation Algorithm

- To summarize the relations to derive the back-propagation algorithm: First, the correction $\Delta w_{ji}(n)$ applied to the synaptic weight connecting neuron i to neuron j is defined by the delta rule:

$$\begin{pmatrix} Weight \\ correction \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} Learning \\ rate\ parameter \\ \eta \end{pmatrix} . \begin{pmatrix} Local\ g \\ radient \\ \delta_j(n) \end{pmatrix} . \begin{pmatrix} Input\ signal \\ of\ neuron\ j \\ y_i(n) \end{pmatrix} \quad (4.25)$$

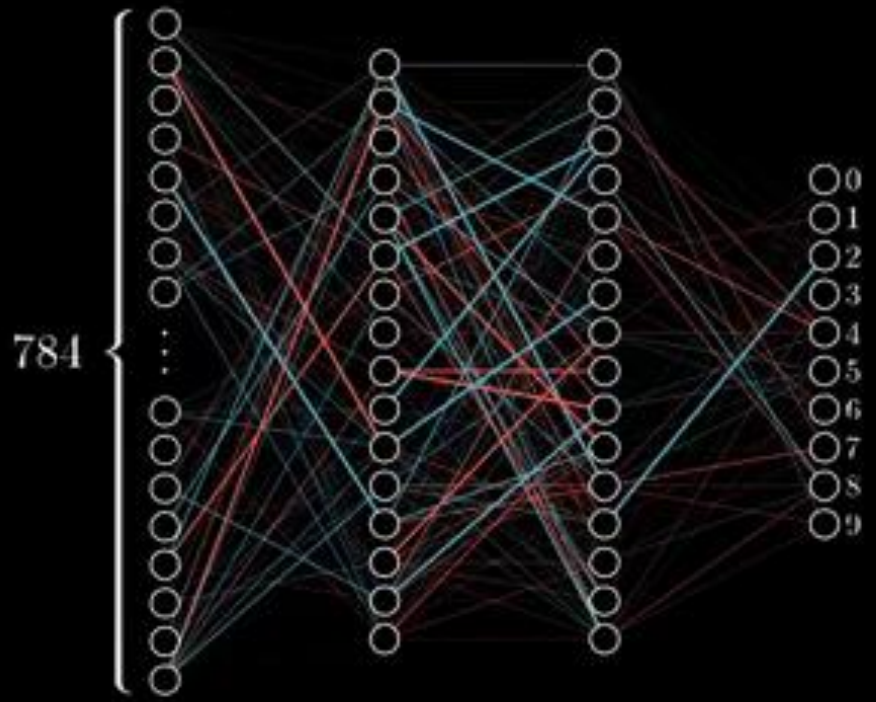
- Second, local gradient $\delta_j(n)$ depends on whether j is output or hidden node

1. If neuron j is an output node, $\delta_j(n)$ equals the product of the derivative $\varphi_j'(v_j(n))$ and error signal $e_j(n)$, both of which are associated with neuron j; see (4.14).

2. If neuron j is a hidden node, $\delta_j(n)$ equals product of associated derivative $\varphi_j'(v_j(n))$ and weighted sum of the δs computed for the neurons in the next hidden or output layer that are connected to neuron i: see (4.24).

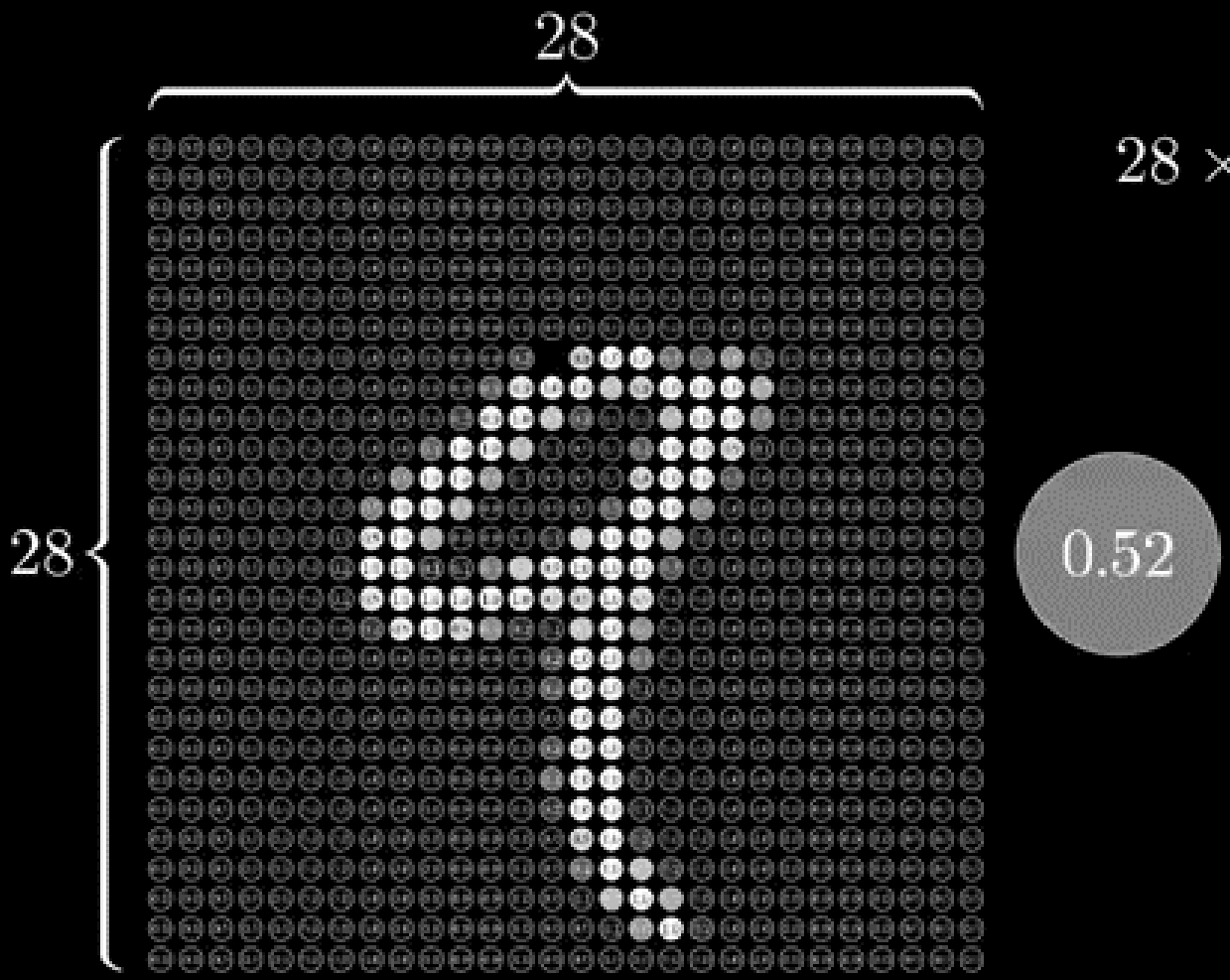
4.3 Back-Propagation Algorithm: Training

Training in progress. . .



https://gfycat.com/helpfulconstantkite-three-blue-one-brown-machine-learning

4.3 Back-Propagation Algorithm: Training



https://gfycat.com/animatedtalkativefunnelweaverspider

The Two Passes of Computation - forward pass and backward pass

- ***The forward pass:*** Synaptic weights remain unaltered throughout the network and the function signals are computed on a neuron-by-neuron basis.
- The function signal appearing at the output of neuron *j* is computed as

$$y_j(n) = \varphi(v_j(n)) \tag{4.26}$$

- where $v_j(n)$ is the induced local field of neuron *j*, defined by

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{4.27}$$

- If neuron *j* is in the first hidden layer of the network, $m = m_0$ and the index *i* refers to the *i*th input terminal of the network, for which we write

$$y_i(n) = x_i(n) \tag{4.28}$$

The Two Passes of Computation - forward pass and backward pass

- If neuron j is in the output layer of the network, $m = m_L$ and the index j refers to the jth output terminal of the network, for which we write

$$y_j(n) = o_j(n) \tag{4.28}$$

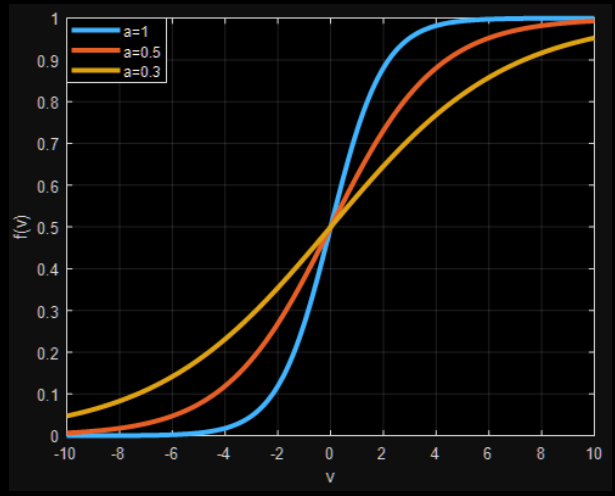
- where $o_j(n)$ is the jth element of the output vector (pattern).

The Two Passes of Computation - forward pass and backward pass

- ***The backward pass:*** Starts at the output layer by passing the error signals leftward through the network, layer by layer and recursively computing the δ (i.e., the local gradient) for each neuron.
- This recursive process permits the synaptic weights of the network to undergo changes in accordance with the delta rule of (4.25).
- For a neuron located in the output layer, the δ is equal to the error signal of that neuron multiplied by first derivative of its nonlinearity.
- Eq. (4.24) is used to compute the δs for all the neurons in the layers and change the weights of all connections feeding into it.
- The changes propagate to all synaptic weights in the network.

4.3 Back-Propagation Algorithm

Activation Function



- Activation function should be continuously differentiable. An example of a continuously differentiable nonlinear activation function commonly used in multilayer perceptrons is the sigmoidal nonlinearity;
- Two forms are described as:

1. *Logistic Function*. This form of sigmoidal nonlinearity is defined by

$$\varphi_j\left(v_j(n)\right) = \frac{1}{1+\exp(-av_j(n))} \qquad a > 0 \text{ and } -\infty < v_j(n) < \infty \qquad (4.30)$$

where $v_j(n)$ is the induced local field of neuron j and $0 \leq y_j \leq 1$.

Differentiating (4.30) w.r.t. $v_j(n)$, we get

$$\varphi_j'\left(v_j(n)\right) = \frac{aexp(-av_j(n)}{\left[1+\exp\left(-av_j(n)\right)\right]^2} \qquad (4.31)$$

4.3 Back-Propagation Algorithm

- With $y_j(n) = \varphi_j\left(v_j(n)\right)$, we have

$$\varphi_j'\left(v_j(n)\right) = a y_j(n)[1 - y_j(n)] \tag{4.32}$$

- When the neuron j is an output node and located at the output and $y_j(n) = o_j(n)$, the gradient for neuron j is

$$\delta_j(n) = e_j(n)\varphi_j'\left(v_j(n)\right) = a\left[d_j(n) - o_j(n)\right]o_j(n)\left[1 - o_j(n)\right] \tag{4.33}$$

- for an arbitrary hidden neuron j, we may Express the local gradient as

$$\delta_j(n) = \varphi_j'\left(v_j(n)\right)\sum_k \delta_k(n)w_{kj}(n) = a y_j(n)[1 - y_j(n)]\sum_k \delta_k(n)w_{kj}(n) \tag{4.34}$$

4.3 Back-Propagation Algorithm

1. *Hyperbolic Tangent Function.* Hyperbolic tangent function, is defined by

$$\varphi_j\left(v_j(n)\right) = a\tanh\left(bv_j(n)\right), \qquad (a,b) > 0\,, \quad a \text{ and } b \text{ constant} \tag{4.35}$$

Its derivative with respect to $v_j(n)$ is given by

$$\varphi_j'\left(v_j(n)\right) = ab\,\mathrm{sech}^2(bv_j(n)) = ab\left(1-\tanh^2(bv_j(n)\right)$$

$$= \frac{b}{a}\left[a - y_j(n)\right]\left[a + y_j(n)\right] \tag{4.36}$$

- For a neuron j located in the output layer, the local gradient is

$$\delta_j(n) = e_j(n)\varphi_j'\left(v_j(n)\right) = \frac{b}{a}\left[d_j(n) - o_j(n)\right]\left[a - o_j(n)\right]\left[1 + o_j(n)\right] \tag{4.37}$$

- For a neuron j hidden layer, the local gradient is

$$\delta_j(n) = \varphi_j'\left(v_j(n)\right)\sum_k \delta_k(n)w_{kj}(n) = \frac{b}{a}\left[a - y_j(n)\right]\left[a + y_j(n)\right]\sum_k \delta_k(n)w_{kj}(n) \tag{4.38}$$

4.3 Back-Propagation Algorithm

Rate of Learning

- The smaller the learning-rate parameter η, the smaller the changes to the synaptic weights in the network will be from one iteration to the next, and the smoother the trajectory in weight space, the slower rate of learning.
- If η is too large in order to speed up the rate of learning, the resulting large changes in the synaptic weights may make the network unstable
- A simple method of increasing rate of learning yet avoiding danger of instability is to modify the delta rule of (4.13) by including a momentum term shown by

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_j(n) \qquad (4.39)$$

- where α is usually a positive number called the *momentum constant* that controls the feedback loop acting around $\Delta w_{ji}(n)$ as shown in Fig. 4.6.

4.3 Back-Propagation Algorithm

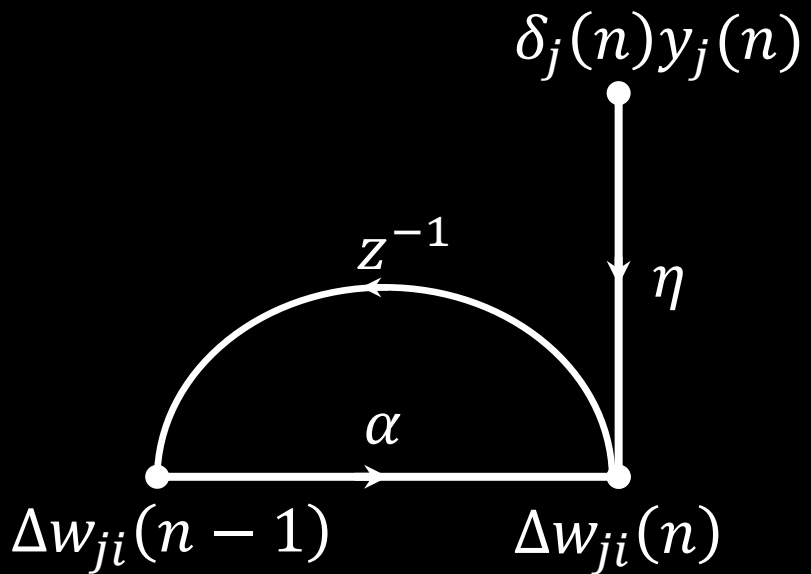


Fig 4.6 Signal-flow graph illustrating the effect of momentum constant *a*.

4.3 Back-Propagation Algorithm

Rate of Learning

- To see the effect the momentum constant a, we rewrite (4.39) as a time series of length n+1 with index t as

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t) \qquad (4.40)$$

- Recalling that $\delta_j(n) y_i(n) = -\partial \mathcal{E} / \partial w_{ji}(n)$, (4.40) can be rewritten as

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial \mathcal{E}(t)}{\partial w_{ji}(t)} \qquad (4.41)$$

4.3 Back-Propagation Algorithm

Rate of Learning

- From $\Delta w_{ji}(n) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial \mathcal{E}(t)}{\partial w_{ji}(t)}$ (4.41), following observations are made

1. The current adjustment $\Delta w_{ji}(n)$ represents the sum of an exponentially weighted time series. For the time series to be convergent $0 \leq \alpha < 1$.
2. When partial derivative $\partial \mathcal{E}(t)/\partial w_{ji}(t)$ has same algebraic sign on consecutive iterations, exponentially weighted sum $\Delta w_{ji}(n)$ grows in magnitude, and so the weight $w_{ji}(n)$ is adjusted by a large amount.

   Inclusion of momentum in the back-propagation algorithm tends to accelerate descent in steady downhill directions.

2. When they have opposite sign, $\Delta w_{ji}(n)$ shrinks in magnitude, and so weight $w_{ji}(n)$ is adjusted by a small amount. Inclusion of momentum in backpropagation algorithm has *stabilizing* effect in directions that oscillate in sign.

4.3 Back-Propagation Algorithm

Sequential and Batch Modes of Training

- In back-propagation algorithm, learning results from the many presentations of a prescribed set of training examples to the multilayer perceptron.
- One complete presentation of the entire training set during the learning process is called an epoch.
- Back-propagation learning may proceed in one of two basic ways:

1. **Sequential Mode**
2. **Batch Mode**

4.3 Back-Propagation Algorithm

1.Sequential Mode

Also referred to as on-line, pattern, or stochastic mode, where operation weight updating is performed after presentation of each training example.

i.ei consider an epoch consisting of N training examples (patterns) arranged as $(\mathbf{x}(1), \mathbf{d}(1)), \ldots, (\mathbf{x}(N), \mathbf{d}(N))$.

The example pairs $(\mathbf{x}(j), \mathbf{d}(j))$ in the epoch are presented to the network and sequence of forward and backward computations are performed sequentially

This results in certain adjustments to the synaptic weights and bias levels of the network.

4.3 Back-Propagation Algorithm

2. Batch Mode

Here, weight updating is performed after the presentation of all the training examples that constitute an epoch.

For a particular epoch, we define the cost function as of (4.2) and (4.3), reproduced here in the composite form:

$$\mathcal{E}_{av} = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in C} e_j^2(n) \qquad (4.42)$$

where the error signal $e_j(n)$ pertains to output neuron j for training example n equals the difference between $d_j(n)$ and $y_j(n)$ which represents the j th element of the desired response vector $\mathbf{d}(n)$

4.3 Back-Propagation Algorithm

Sequential and Batch Modes of Training

- For a learning-rate parameter η, the adjustment applied to synaptic weight w_{ji} connecting neuron i to neuron j, is defined by the delta rule

$$\Delta w_{ji} = -\eta \frac{\partial \mathcal{E}_{av}}{\partial w_{ji}} = \frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\partial e_j(n)}{\partial w_{ji}} \qquad (4.43)$$

- The sequential mode of training is preferred over the batch mode because it requires less local storage for each synaptic connection.

4.3 Back-Propagation Algorithm

- In summary, despite the fact that the sequential mode of back-propagation learning has several disadvantages, it is highly popular (particularly for solving pattern classification problems) for two important practical reasons:

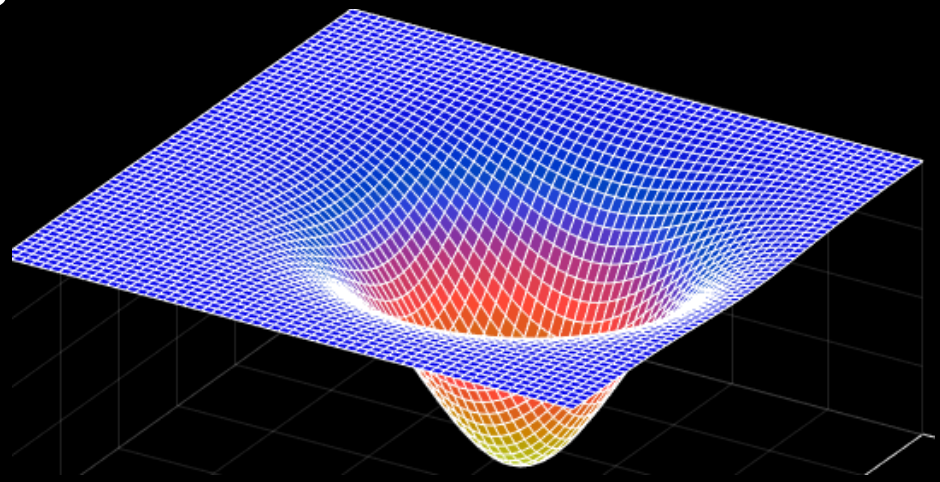
  1. The algorithm is simple to implement.

  2. It provides effective solutions to large and difficult problems.

4.3 Back-Propagation Algorithm

Stopping Criteria

- In general, the back-propagation algorithm cannot be shown to converge, and there are no well-defined criteria for stopping its operation.
- To formulate such a criterion, it is logical to think in terms of the unique properties of a local or global minimum of the error surface.
- A necessary condition for the weight vector $\mathbf{w}^*$ to be a minimum is that the gradient vector $\mathbf{g}(\mathbf{w})$ (i.e., first-order partial derivative) of the error surface with respect to the weight vector $\mathbf{w}$ be zero at $\mathbf{w} = \mathbf{w}^*$.



4.3 Back-Propagation Algorithm

- We may formulate a sensible convergence criterion for back-propagation learning as

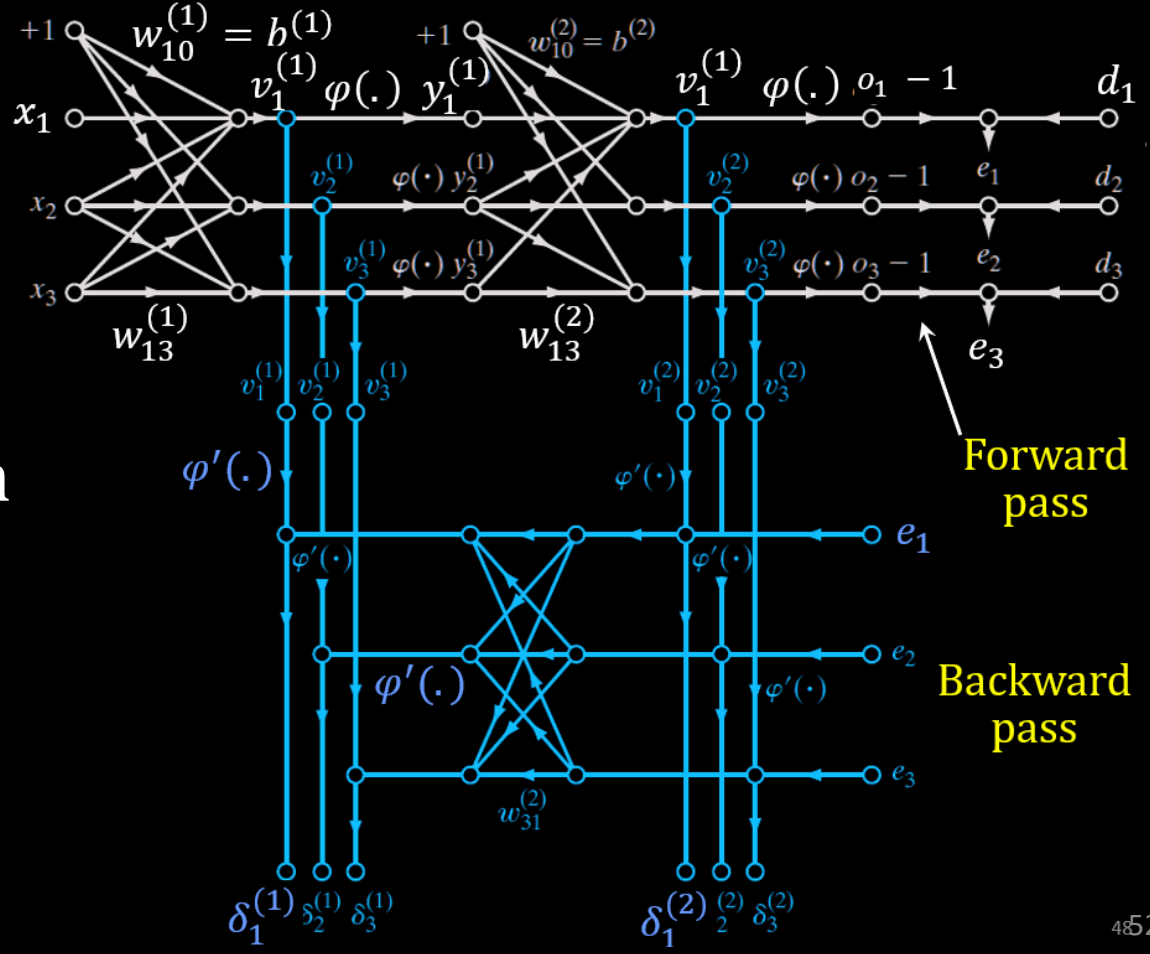
The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.

- Due to the fact that, for successful trials, learning times
- may be long and it requires the computation of the gradient vector $\mathbf{g}(\mathbf{w})$.
- Thus, we can use the stationarity of the cost function or error measure $\mathcal{E}_{av}(w)$ at the point $\mathbf{w} = \mathbf{w}^*$ and suggest a different criterion of convergence:

The back-propagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small.

4.4 Summary of the Back-propagation Algorithm

- The signal-flow graph for *back-propagation learning*, incorporating both the forward and backward phases of the computations involved in the learning process, is presented in Fig. 4.7 for the case of $L = 2$ and $m_0 = m_1 = m_2 = 3$.
- The top part of the signal-flow graph accounts for the forward pass.
- The lower part of the signal-flow graph accounts for the backward pass,
- which is referred to as a sensitivity graph for computing the local gradients in the back-propagation algorithm.



Back-propagation Algorithm

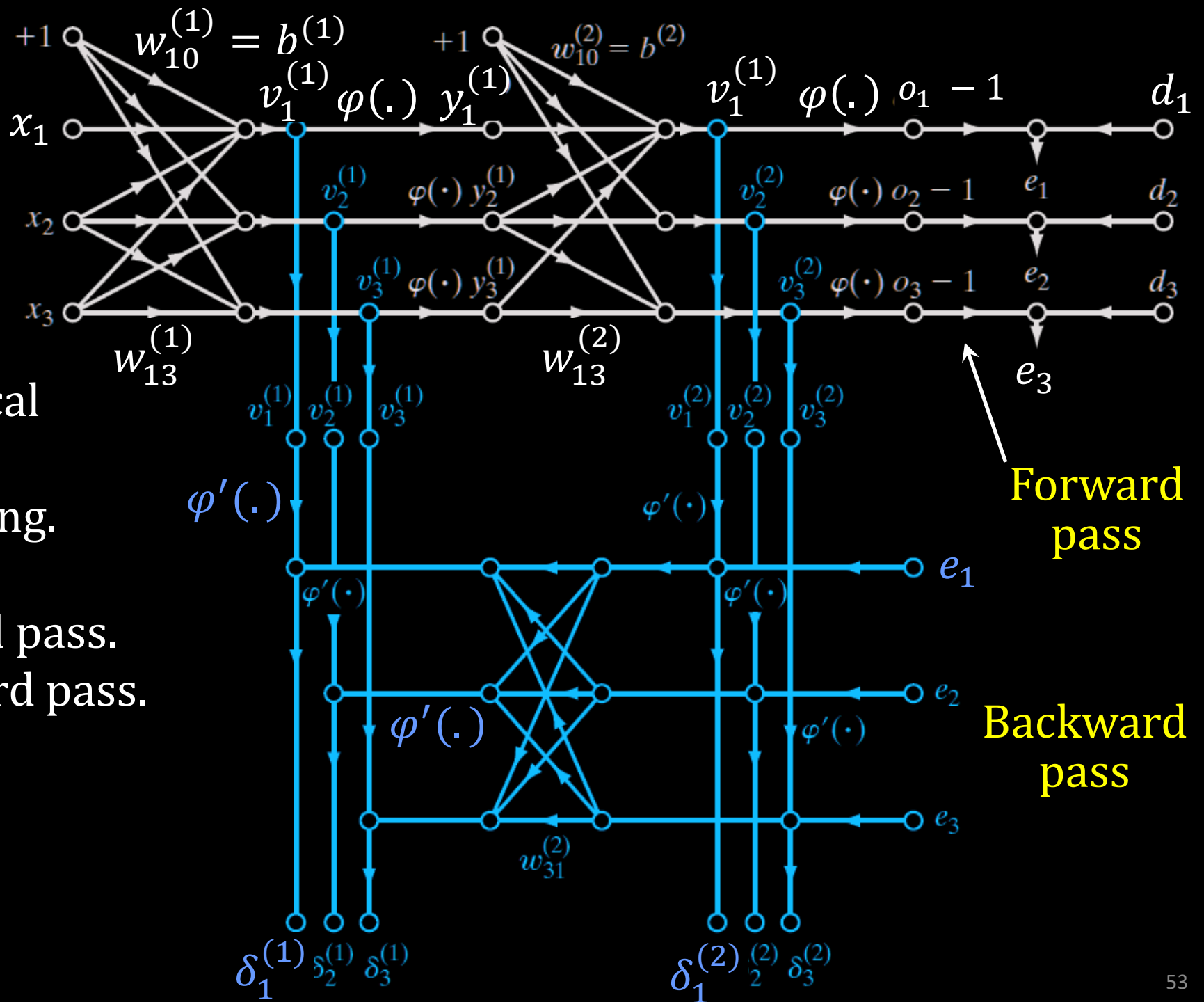


Fig. 4.7 Signal-flow graphical summary of back-propagation learning.

Top part of graph : Forward pass.
Bottom part : Backward pass.

4.4 Summary of the Back-propagation Algorithm

- Sequential updating of weights is the preferred for real-time implementation of the back-propagation algorithm, where operation and the algorithm cycles through the training sample$\{(\mathbf{x}(n), \mathbf{d}(n)\}_{n=1}^{N}$ are summarized as follows:

1. ***Initialization***. Assuming that no prior information is available, pick the synaptic weights and thresholds from a uniform distribution with zero mean and a proper variance.

2. ***Presentations of Training Examples***. Present the network with an epoch of training examples.

4.4 Summary of the Back-propagation Algorithm

3. Forward Computation. Let a training example in the epoch be denoted by $(\mathbf{x}(n), \mathbf{d}(n))$, with the input vector $\mathbf{x}(n)$ applied to the input layer of sensory nodes and the desired response vector $\mathbf{d}(n)$ presented to output layer of computation nodes.

Compute induced local fields and function signals of the network by proceeding forward through the network, layer-by-layer. The induced local field $v_j^l(n)$ for neuron j in layer l is

$$v_j^l(n) = \sum_{i=0}^{m_0} w_{ji}^{(l)}(n) y_i^{(l-1)}(n) \qquad (4.44)$$

whe.re $y_i^{(l-1)}(n)$ is the output (function) signal of neuron i in the previous layer l-1 at iteration n and $w_{ji}^{(l)}$ is the synaptic weight of neuron j in layer l that is fed from neuron i in layer l-1.

4.4 Summary of the Back-propagation Algorithm

- If neuron j is in the output layer (i.e., $l = L$, where L is the network depth, set

$$y_j^{(L)} = o_j(n)$$

- The error signal can then be computed as

$$e_j(n) = d_j(n) - o_j(n) \qquad (4.45)$$

Where $d_j(n)$is the jth element of the desired response vector $\mathbf{d}(n)$.

4.4 Summary of the Back-propagation Algorithm

4. *Backward Computation*. Compute the δs (i.e., local gradients) of the network, defined by

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(l)}(n)\varphi_j'\left(v_j^{(L)}(n)\right) & \text{for neuron } j \text{ in output layer } L \\ \varphi_j'\left(v_j^{(L)}(n)\right)\sum_k \delta_k^{(l+1)}(n)\, w_{kj}^{(l+1)}(n) & \text{for neuron } j \text{ in hidden layer } l \end{cases} \quad (4.46)$$

where the prime in $\varphi_j'(.)$ denotes differentiation with respect to the argument.

- Adjust the synaptic weights of the network in layer l according to the generalized delta rule:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[w_{ji}^{(l)}(n-1) + \eta\delta_j^{(l)}(n)y_i^{(l-1)}(n)] \quad (4.47)$$

Where η is the learning rate parameter and α is the momentum constant.

4.4 Summary of the Back-propagation Algorithm

5. ***Iteration.*** Iterate the forward and backward computations under points 3 and 4 by presenting new epochs of training examples randomly to the network until the stopping criterion is met.

End of Section 4.1 - 4.4