Nonlinear regression methods

Neural networks

Michela Mulas

A quick recap

During the last lectures, we did...

- **Neural network models for regression**.

  Neural networks are powerful nonlinear regression techniques inspired by theories about how the brain works.

  Some useful properties and capabilities of neural network are:

  - ⇝ **Nonlinearity**: It is an important property, particularly if the underlying physical mechanisms responsible for generation of the input signal is inherently nonlinear.
  - ⇝ **Input-output mapping**: The supervised learning involves modification of the synaptic weights by applying a set of training examples.

    Each example consists of a unique input signal and a corresponding desired (target) response.

    The training of the network is repeated for many examples in the set, until the network reaches a steady state where there are no further significant changes in the synaptic weights.

A quick recap

During the last lectures, we did...

- **Neural network models for regression**.

  Neural networks are powerful nonlinear regression techniques inspired by theories about how the brain works.

  Some useful properties and capabilities of neural network are:

  - ⇝ **Adaptivity**: A neural network trained to operate in a specific environment can be easily retrained to deal with minor changes in the operating environmental.

    When it is operating in a nonstationary environment (i.e., one where statistics change with time), a neural network may be designed to change its synaptic weights in real time.
  - ⇝ **Fault Tolerance**: A neural network, implemented in hardware form, has the potential to be inherently fault tolerant, or capable of robust computation, in the sense that its performance degrades gracefully under adverse operating conditions.

A quick recap

During the last lectures, we did...

- **Neural network models for regression**.

  The outcome is modeled by an intermediary set of unobserved variables.

  - ⇝ Called hidden variables or hidden units.

  Each hidden unit is a linear combination of some or all of the predictor variables.

  This linear combination is typically transformed by a nonlinear function g(·), such as the logistic (i.e., sigmoidal) function.

$$h_k(\mathbf{x}) = g\left(\beta_{0k} + \sum_{j=1}^{P} x_j \beta_{jk}\right)$$

$$g(u) = \frac{1}{1 + e^{-u}}$$

  The β coefficients are similar to the regression coefficients.

  β_{jk} is the effect of the jth predictor on the kth hidden unit.

A quick recap

During the last lectures, we did...

- **Neural network models for regression**.

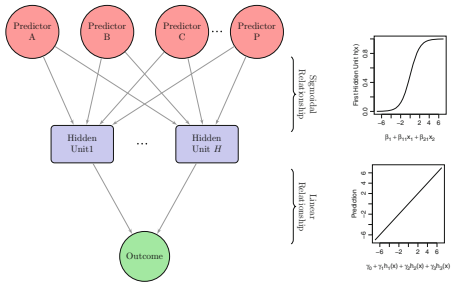


Diagram of a neural network with a single hidden layer.

The hidden units are linear combinations of the predictors that have been transformed by a sigmoidal function.

A quick recap

During the last lectures, we did...

- **Neural network models for regression**.

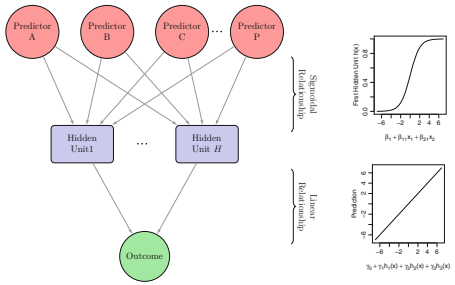


Diagram of a neural network with a single hidden layer.

The output is modeled by a linear combination of the hidden units

A quick recap

During the last lectures, we did...

- **Neural network models for regression**.

  A neural network model usually involves multiple hidden units to model the outcome.

  The number of hidden units must be defined.

  Each unit must be then related to the outcome.

  Another linear combination connects the hidden units to the outcome:

$$f(\mathbf{x}) = \gamma_0 + \sum_{k=1}^{H} \gamma_k h_k$$

  For this type of network model and P predictors:

  - ⇝ There are a total of $H(P+1)+H+1$ parameters being estimated.
  - ⇝ Quickly becomes large as P increases.

A quick recap

During the last lectures, we did...

- **Neural network models for regression**.

  Treating this model as a **nonlinear regression model**, the parameters are usually optimized to **minimize the sum of the squared residuals**.

  This can be a challenging numerical optimization problem (there are no constraints on the parameters of this complex nonlinear model).

  The parameters are usually initialized to random values and then specialized algorithms for solving the equations are used.

  The **back-propagation algorithm** is a highly efficient methodology that works with derivatives to find the optimal parameters.

Today's goal

Today, we going to continue with...

- **Neural network models for regression**.
- More on back-propagation algorithm
  The two steps of procedure
  Examples in R

Reading list

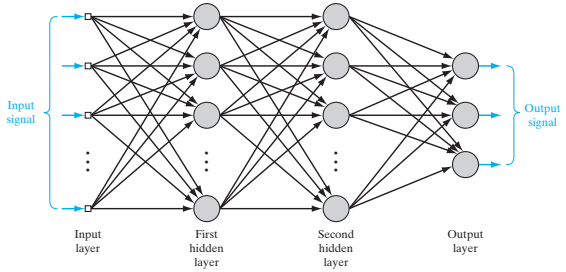
Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*, Springer (2014)

Simon Haykin. *Neural Networks: A Comprehensive Foundation*[1], Pearson (2008, 3rd edition)

[1] Note that we will follow its notation for the rest of the lecture.

Multi-layer perceptrons

Consider a multiplayer perceptron with two hidden layers and an output layer.

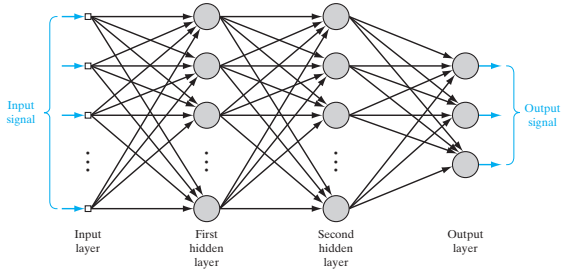


Basic features of multilayer perceptrons are:

- Each neuron model includes a **nonlinear activation function** that is **differentiable**.
- There are **one or more layers hidden** from both the input and output nodes.
- The network exhibits a **high degree of connectivity**, the extent of which is determined by synaptic weights of the network.

Multi-layer perceptrons

Consider a multiplayer perceptron with two hidden layers and an output layer.

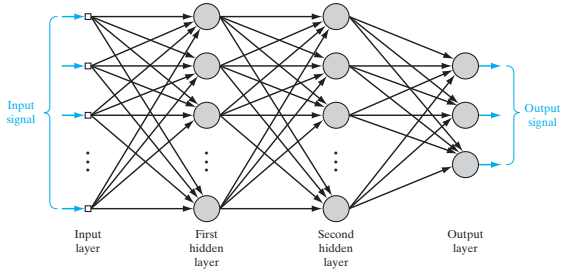


This network shown is **fully connected**: A neuron in any layer of the network is connected to all the neurons (nodes) in the previous layer.

Signal flow through the network progresses in a **forward direction**, from left to right and on a layer-by-layer basis.

Multi-layer perceptrons

Consider a multiplayer perceptron with two hidden layers and an output layer.



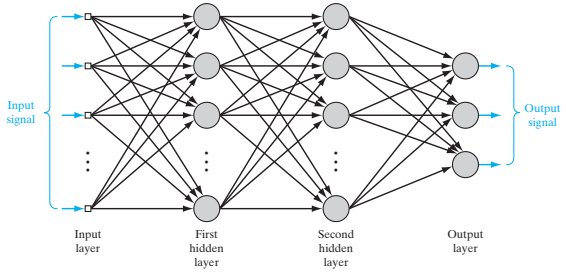
Two kinds of signals are identified:

- **Input signal**, also called function signal, is the signal (stimulus) that comes in at the input end of the network.

  It propagates forward (neuron by neuron) through the network, and emerges at the output end of the network as an output signal.

Multi-layer perceptrons

Consider a multiplayer perceptron with two hidden layers and an output layer.



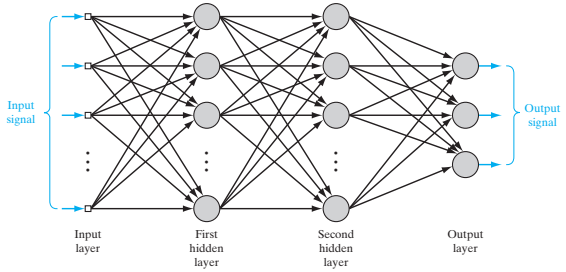
Two kinds of signals are identified:

- **Error signal**: originates at an output neuron of the network and propagates backward (layer by layer) through the network.

  We refer to it as an "error signal" because its computation by every neuron of the network involves an error-dependent function in one form or another.

Multi-layer perceptrons

Consider a multiplayer perceptron with two hidden layers and an output layer.



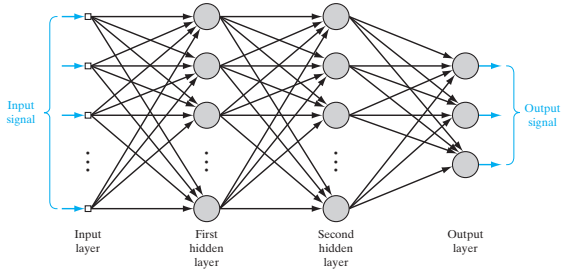
The output neurons constitute the **output layer** of the network.

The remaining neurons constitute **hidden layers** of the network.

- ⇝ The hidden units are not part of the output or input of the network.
- ⇝ The first hidden layer is fed from the input layer made up of sensory units (source nodes); the resulting outputs of the first hidden layer are in turn applied to the next hidden layer; and so on for the rest of the network.

Multi-layer perceptrons

Consider a multiplayer perceptron with two hidden layers and an output layer.

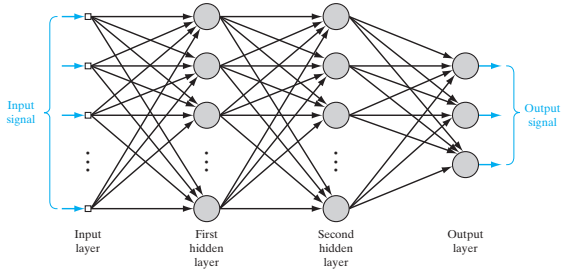


Each hidden or output neuron is designed to perform two computations:

- ⇝ The computation of the function signal appearing at the output of each neuron (expressed as a continuous nonlinear function of the input signal and synaptic weights associated with that neuron)
- ⇝ 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.

Multi-layer perceptrons

Consider a multiplayer perceptron with two hidden layers and an output layer.

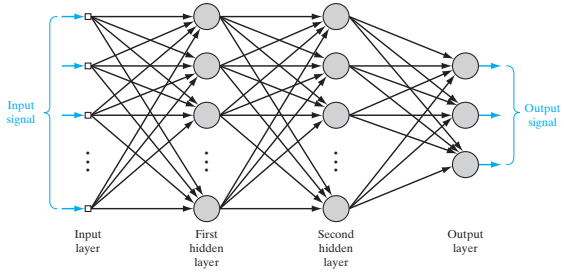


The hidden neurons act as **feature detectors**.

- ⇝ As the learning process progresses across the multilayer perceptron, the hidden neurons begin to gradually "discover" the salient features that characterize the training data.
- ⇝ They do so by performing a nonlinear transformation on the input data into a new space called the feature space.

Multi-layer perceptrons

Consider a multiplayer perceptron with two hidden layers and an output layer.

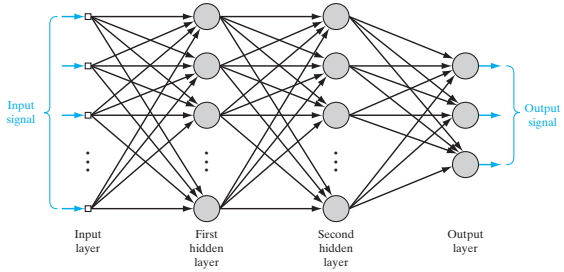


The supervised learning can be **batch**:

- ⇝ Adjustments to the synaptic weights are performed after the presentation of all the examples N in the training sample that constitute one epoch of training.
- ⇝ The cost function for batch learning is defined by the average error energy $\mathscr{E}_{av}$
- ⇝ Adjustments to the synaptic weights of the multilayer perceptron are made on an **epoch-by-epoch basis**.

Multi-layer perceptrons

Consider a multiplayer perceptron with two hidden layers and an output layer.

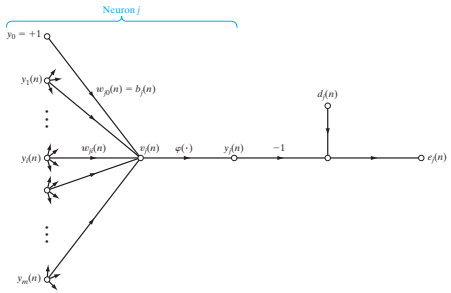


The supervised learning can be **on-line**:

- ⇝ Adjustments to the synaptic weights of the multilayer perceptron are performed on an **example-by-example basis**.
- ⇝ The cost function to be minimized is the total instantaneous error energy $\mathscr{E}$.

Back-propagation algorithm

To describe the back-propagation algorithm, consider a neuron j being fed by a set of function signals produced by a layer of neurons to its left.



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

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

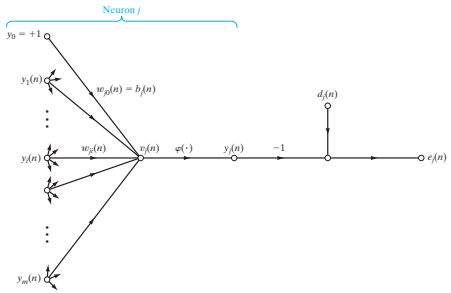
m is the total number of inputs (excluding the bias) applied to the neuron j.

- The function signal $y_j(n)$ at the output of the neuron j at iteration n is

$$y_j(n) = \varphi_j(v_j(n))$$

Back-propagation algorithm

To describe the back-propagation algorithm, consider a neuron j being fed by a set of function signals produced by a layer of neurons to its left.



- The back-propagation applies a correction $\Delta w_{ji}(n)$ to the synaptic weights $w_{ji}(n)$:

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

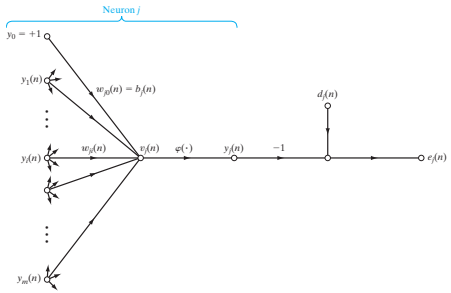
This is known as the **delta-rule**.

η is the **learning-rate** parameter.

The minus sign accounts for gradient descent in weight space (i.e., seeking a direction for weight change that reduces the value of $\mathscr{E}(n)$).

Back-propagation algorithm

To describe the back-propagation algorithm, consider a neuron j being fed by a set of function signals produced by a layer of neurons to its left.



- The partial derivative of **total instantaneous error energy** of the whole network $\mathscr{E}(n)$ can be found using the **chain rule** of calculus:

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

$\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)}$ represents a **sensitivity factor**.

It gives the direction of search in weight space for the synaptic weight.

Back-propagation algorithm

To describe the back-propagation algorithm, consider a neuron j being fed by a set of function signals produced by a layer of neurons to its left.

- Differentiating $\mathscr{E}(n)$ with respect to the **error signal** gives

$$\frac{\partial \mathscr{E}(n)}{\partial e_j(n)} = \frac{\partial}{\partial e_j(n)} \left(\frac{1}{2} \sum_{j \in C} e_j^2 \right) = e_j(n)$$

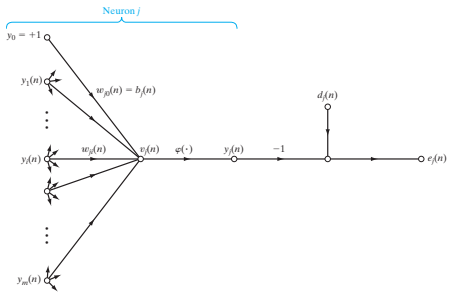
C has all the neurons in the output layer.

$e_j(n) = d_j(n) - y_j(n)$

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

Back-propagation algorithm

To describe the back-propagation algorithm, consider a neuron j being fed by a set of function signals produced by a layer of neurons to its left.



- Differentiating $e_j(n)$ with respect to $y_j(n)$

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

- Differentiating $y_j(n)$ with respect to $v_j(n)$

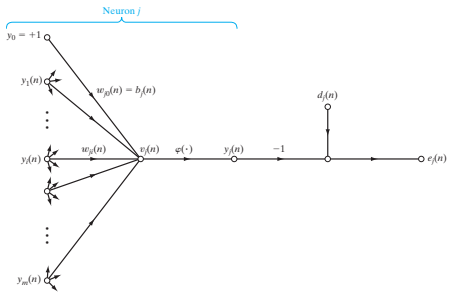
$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'(v_j(n))$$

- Differentiating $v_j(n)$ with respect to $w_{ji}(n)$

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

Back-propagation algorithm

To describe the back-propagation algorithm, consider a neuron j being fed by a set of function signals produced by a layer of neurons to its left.



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

$$\Delta w_{ji}(n) = -\eta \underbrace{e_j(n)\varphi_j'(v_j(n))}_{\text{local gradient } \delta_j(n)} y_i(n)$$

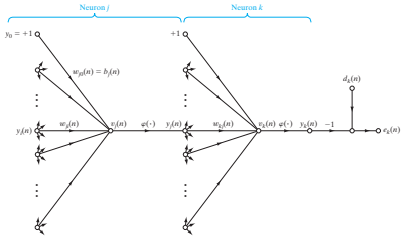
The local gradient $\delta_j(n)$ is defined by:

$$\begin{aligned} \delta_j(n) &= \frac{\partial \mathscr{E}(n)}{\partial v_j(n)} = \frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= e_j(n)\varphi'(v_j(n)) \end{aligned}$$

It points to required changes in synaptic weights.

Back-propagation algorithm

A key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j.

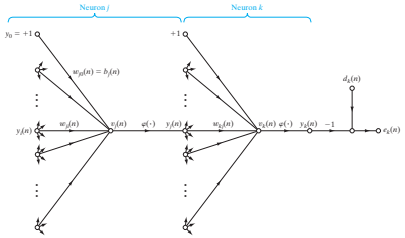


Case 1 Neuron j is an **output node**

- When neuron j is located in the output layer of the network, it is supplied with a desired response of its own.
  - ⇝ We use $d_j(n)$ to compute the error signal $e_j(n)$ associated with this neuron.
  - ⇝ Then, it is straightforward to compute the local gradient $\delta_j(n)$.

Back-propagation algorithm

A key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j.



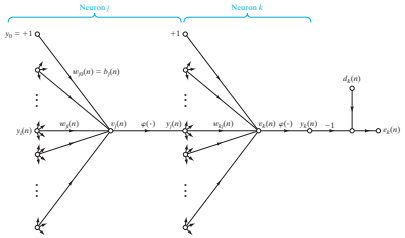
Case 1 Neuron j is a **hidden node**

- When neuron j is located in a hidden layer of the network, there is no specified desired response for that neuron.

  Accordingly, the error signal for a hidden neuron would have to be determined recursively and working backwards in terms of the error signals of all the neurons to which that hidden neuron is directly connected.

Back-propagation algorithm

A key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j.



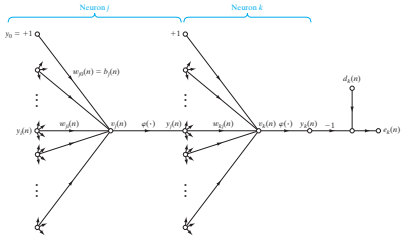
Case 1 Neuron j is a hidden node

- We may redefine the local gradient $\delta_j(n)$ for **hidden neuron** j as:

$$\delta_j(n) = -\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \varphi'(v_j(n))$$

Back-propagation algorithm

A key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j.



Case 1 Neuron j is a hidden node

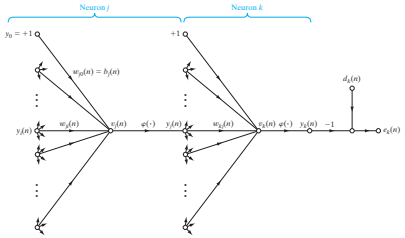
- To calculate the first partial derivative, we see that

$$\mathscr{E}(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n) \quad \text{neuron } k \text{ is an output layer}$$

$$\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$

Back-propagation algorithm

A key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j.



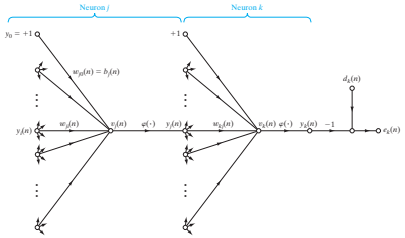
Case 1 Neuron j is a hidden node

- Using the chain rule and noting that $e_k(n) = d_k - y_k(n) = d_k - \varphi_k(v_k(n))$, we have

$$\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} = -\sum_k \underbrace{e_k(n)\varphi'(v_k(n))}_{\text{local gradient } \delta_k} w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n)$$

Back-propagation algorithm

A key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j.



Case 1 Neuron j is a hidden node

- We get the back-propagation formula for the local gradient $\delta_j(n)$, described by

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

Back-propagation algorithm

Summarizing, the relations that we have derived for the back-propagation algorithm:

1. The correction $\Delta w_{ji}(n)$ applied to the synaptic weight connecting neuron i to neuron j is defined by the delta rule:

$$\Big(\text{Weight correction, } \Delta w_{ji}(n)\Big) = \Big(\text{Learning rate, } \eta\Big) \times \Big(\text{Local gradient, } \delta_j(n)\Big) \times \Big(\text{Input signal to neuron } j, y_i\Big)$$

2. The local gradient $\delta_j(n)$ depends on whether neuron j is an output node or a hidden node:
   - ⇝ **Output node**: $\delta_j(n)$ equals the product of the derivative $\varphi_j'(v_j(n))$ and the error signal $e_j(n)$, both of which are associated with neuron j.
   - ⇝ **Hidden node**: $\delta_j(n)$ equals the product of the associated derivative $\varphi_j'(v_j(n))$ and the weighted sum of the δs computed for the neuron in the next hidden layer or output layer that are connected to neuron j.

Back-propagation algorithm

The two passes of computation

In the application of the back-propagation algorithm, we have:

1. In the **forward phase**, the synaptic weights of the network are fixed and the input signal is propagated through the network, layer by layer, until it reaches the output.

   The function signal appearing at the output of neuron j is computed as

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

   $v_j(n)$ is the induced local field of the neuron j.

   m is the total number of inputs (excluding the bias) applied to neuron j.

   w_{ji} is the synaptic weight connecting neuron i to neuron j.

   $y_i(n)$ is an input signal of neuron j, or, equivalently, the function signal appearing at the output of neuron i.

Back-propagation algorithm
The two passes of computation

In the application of the back-propagation algorithm, we have:

2. In the **backward phase**, 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.

   For a neuron located in the output layer, the δ is simply equal to the error signal of that neuron multiplied by the first derivative of its nonlinearity.

   The recursive computation is continued, layer by layer, by propagating the changes to all synaptic weights in the network.

Back-propagation algorithm
The rate of learning

The back-propagation algorithm provides an "approximation" to the trajectory in weight space computed by the method of steepest descent.

The learning-rate parameter η plays an important role:

- A smaller η means smaller the changes to the synaptic weights in the network from one iteration to the next and smoother trajectory in weight space.
  - $\rightsquigarrow$ We get a slower rate of learning.
- Big ηs result in large changes in the synaptic weights and in a possibly unstable (i.e., oscillatory) network.
  - $\rightsquigarrow$ We speed up the rate of learning.

To increasing the rate of learning while avoiding instability, we modify the delta rule by including a momentum term:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \quad \textbf{Generalized delta rule}$$

α is called the momentum constant.

It controls the the feed- back loop acting around Δw_{ji}

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.

Rather, there are some **reasonable criteria**, each with its own practical merit, that may be used to terminate the weight adjustments.

1. Let the weight vector $\mathbf{w}^*$ denote a minimum, be it local or global.

   A necessary condition for $\mathbf{w}$ to be a minimum is that the gradient vector $\mathbf{g}(\mathbf{w})$ of the error surface with respect to the weight vector w must be zero at $\mathbf{w} = \mathbf{w}^*$.

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

   Drawback: Learning times may be long for successful trials. It also requires the computation of the gradient vector $\mathbf{g}(\mathbf{w})$.

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.

Rather, there are some **reasonable criteria**, each with its own practical merit, that may be used to terminate the weight adjustments.

2. Another unique property of a minimum that we can use is the fact that the cost function $\mathscr{E}_{av}(\mathbf{w})$ is stationary at the point $\mathbf{w} = \mathbf{w}^*$.

   $\rightsquigarrow$ 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.

   Drawback: It may result in a premature termination of the learning process.

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.

Rather, there are some **reasonable criteria**, each with its own practical merit, that may be used to terminate the weight adjustments.

3. After each learning iteration, the network is tested for its generalization performance.

   ⇝ The learning process is stopped when the generalization performance is adequate or when it is apparent that the generalization performance has peaked