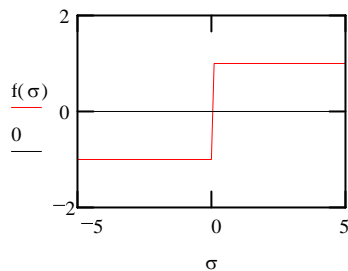


### Each neuron in a layered neural network...

- Has either binary or real inputs.  $x_j$  for  $j=\{0, 1, \dots, m\}$  where  $j=0$  is the bias/threshold with  $x_0=1$  and  $m$  is the number of independent variables.
- Sums up weighted inputs.  $\sigma = \sum w_j \cdot x_j$  for  $j=\{0, 1, \dots, m\}$ .
- Has an output signal level given by  $f(\sigma)$ . Some common functions are listed below.

Step function (perceptrons)

$$f(\sigma) := \text{if}(\sigma \geq 0, 1, 0) \quad \text{or} \quad f(\sigma) := \text{if}(\sigma \geq 0, 1, -1) \quad \sigma := -5, -4.9 \dots 5$$



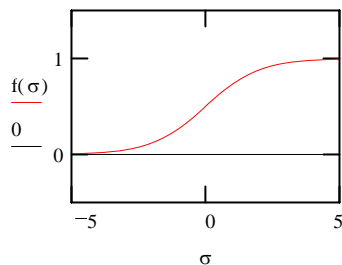
Special properties:

Binary outputs: either -1 or 1

$$\frac{d}{d\sigma} f = 0 \quad \text{except at } \sigma = 0$$

Sigmoid (sigma) function

$$f(\sigma) := \frac{1}{1 + e^{-\sigma}}$$



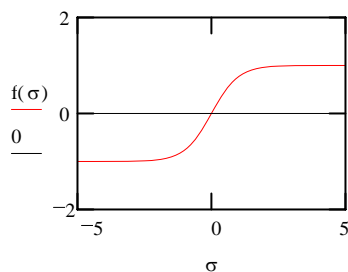
Special properties:

Range from 0 to 1

$$\frac{d}{d\sigma} f = f \cdot (1 - f)$$

Sigmoid (hyperbolic tangent) function

$$f(\sigma) := \tanh(\sigma)$$



Special properties:

Range from -1 to 1

$$\frac{d}{d\sigma} f = 1 - f^2 = (1 + f) \cdot (1 - f)$$

Since  $f$  is calculated when going forward from the input layer to the output layer, this information is saved to calculate  $df/d\sigma$  directly.

The most common feed-forward neural network architecture has three layers: an input layer consisting of  $m$  noncomputing neurons, a hidden layer consisting of  $p$  computing neurons, and an output layer consisting of  $q$  computing neurons. The weights associated with the connection from the  $j$ th neuron in the input layer to the  $k$ th neuron in the hidden layer are denoted as  $w_{kj}$ , and the weights associated with the connection from the  $k$ th neuron in the hidden layer to the  $l$ th neuron in the output layer are denoted as  $v_{lk}$ . Output from the  $k$ th hidden neuron is:

$$z_k = f\left(\sum_{j=0}^m w_{kj} \cdot x_j\right) = f(w_k \cdot x) \quad \text{for } k=1, \dots, p \quad \begin{array}{l} x_0=1 \quad \text{for } j=0 \text{ (bias to the hidden neurons)} \\ z_0=1 \quad \text{for } k=0 \text{ (bias to the output neurons)} \end{array}$$

Output from the  $l$ th output neuron is:

$$O_l = f\left(\sum_{k=0}^p v_{lk} \cdot z_k\right) = f(v_l \cdot z) \quad \text{for } l=1, \dots, q$$

**Training Procedure (Steepest Descent Algorithm).** We can derive the training procedure by minimizing the error function over all  $i$  training patterns and  $l$  output variables  $y$ .

$$\text{Minimize}_{w, v} \text{sse}(w, v) = E(w, v) = \sum_{i=0}^N \sum_{l=1}^q [(\text{error}^{\langle l \rangle})_i]^2 = \sum_{i=0}^N \sum_{l=1}^q [(y^{\langle l \rangle})_i - f(v_l \cdot z)_i]^2$$

To find the expression for updating  $v$ , we take derivative of sse w.r.t.  $v$  by the chain rule.

$$\begin{aligned} \frac{d}{dv_l} \text{sse} &= \sum_{i=0}^N \sum_{l=1}^q \frac{d}{d[(y^{\langle l \rangle})_i - f(v_l \cdot z)_i]} [(y^{\langle l \rangle})_i - f(v_l \cdot z)_i]^2 \cdot \frac{d}{df} [(y^{\langle l \rangle})_i - f(v_l \cdot z)_i] \cdot \left[ \frac{d}{d(v_l \cdot z)_i} f(v_l \cdot z)_i \right] \\ &= -2 \cdot \left[ \sum_{i=0}^N \sum_{l=1}^q [(y^{\langle l \rangle})_i - f(v_l \cdot z)_i] \cdot f'_i \cdot (1 - f'_i) \cdot z_i \right] \\ &= -2 \cdot \left[ \sum_{i=0}^N \sum_{l=1}^q [(y^{\langle l \rangle})_i - (O^{\langle l \rangle})_i] \cdot (O^{\langle l \rangle})_i \cdot [1 - (O^{\langle l \rangle})_i] \cdot z_i \right] \end{aligned}$$

Gradient of error w.r.t. weights  $v$ :

$$\Delta E_v = -2 \cdot \sum_{i=0}^N \sum_{l=1}^q [(y^{\langle l \rangle})_i - (O^{\langle l \rangle})_i] \cdot (O^{\langle l \rangle})_i \cdot [1 - (O^{\langle l \rangle})_i] \cdot z_i$$

In the method of steepest descent, we update the weights  $v$  in a direction opposite to the gradient of error w.r.t.  $v$  with a relative step size of  $\varepsilon$ . The variable  $\varepsilon$  is called the **learning rate**. We can start with a small value of  $\varepsilon$  first and increase it near a local minimum. We use the above gradient when we update the weights after the entire batch has been presented. The following is the updating formula after presenting each training sample sequentially. (We eliminate the summation over  $i$  because we update the formula after the  $i$ th sample; we eliminate the summation over  $l$  because we are only expressing the change in  $v_l$ .)

$$\Delta v_l = \varepsilon \cdot (y^{\langle l \rangle} - f(v_l \cdot z)) \cdot f(v_l \cdot z) \cdot (1 - f(v_l \cdot z)) \cdot z = \varepsilon \cdot (y^{\langle l \rangle} - O^{\langle l \rangle}) \cdot O^{\langle l \rangle} \cdot (1 - O^{\langle l \rangle}) \cdot z$$

Equivalent element-wise notation:

$$\Delta v_{lk} = \varepsilon \cdot (y^{<1>} - O^{<1>}) \cdot O^{<1>} \cdot (1 - O^{<1>}) \cdot z_k$$

Note that we can easily evaluate the derivative of  $f$ . In numerical computation with a von Neuman computer, we can save some computational effort by saving the intermediate results.

$$\delta_l = (y^{<1>} - O^{<1>}) \cdot O^{<1>} \cdot (1 - O^{<1>})$$

$$\Delta v_{lk} = \varepsilon \cdot \delta_l \cdot z_k$$

Expressed in an iteration form:

$$(v^{<new>})_{lk} = (v^{<old>})_{lk} + \varepsilon \cdot \delta_l \cdot z_k$$

We can also add a momentum term to speed up convergence.

$$(v^{<new>})_{lk} = (v^{<old>})_{lk} + \varepsilon \cdot \delta_l \cdot z_k + M \cdot (\Delta v_{lk})^{<old>}$$

Similarly, to find the expression for updating  $w$ , we take derivative of sse w.r.t.  $w$  by the chain rule.

$$\frac{d}{dw_{kj}} \text{sse} = \sum_{k=1}^p \frac{d}{dz_k} \text{sse} \cdot \frac{d}{dw_{kj}} z_k$$

where

$$\frac{d}{dz_k} \text{sse} = -2 \cdot \left[ \sum_{i=0}^N \sum_{l=1}^q [(y^{<1>})_i - (O^{<1>})_i] \cdot (O^{<1>})_i \cdot [1 - (O^{<1>})_i] \cdot v_{lk} \right]$$

$$\frac{d}{dw_{kj}} z_k = \sum w_{kj} \cdot \frac{d}{dx_{ij}} f(\sum w_{kj} \cdot x_{ij}) \cdot \frac{d}{dw_{kj}} \sum w_{kj} \cdot x_{ij} = z_{ik} \cdot (1 - z_{ik}) \cdot x_{ij}$$

Thus,

$$\frac{d}{dw_{kj}} \text{sse} = \sum_{k=1}^p -2 \cdot \left[ \sum_{i=0}^N \sum_{l=1}^q [(y^{<1>})_i - (O^{<1>})_i] \cdot (O^{<1>})_i \cdot [1 - (O^{<1>})_i] \cdot v_{lk} \right] \cdot z_{ik} \cdot (1 - z_{ik}) \cdot x_{ij}$$

Gradient of error w.r.t. weights  $w$ :

$$\Delta E_w = -2 \cdot \sum_{k=1}^p \left[ \sum_{i=0}^N \sum_{l=1}^q [(y^{<1>})_i - (O^{<1>})_i] \cdot (O^{<1>})_i \cdot [1 - (O^{<1>})_i] \cdot v_{lk} \right] \cdot z_{ik} \cdot (1 - z_{ik}) \cdot x_{ij}$$

The following is the steepest descent algorithm that searches in the negative gradient direction of error for updating the  $j$ th weight after sequentially presenting each example  $i$ . (We eliminate summation over  $i$  for updating after the  $i$ th sample; we eliminate the summation over  $k$  because we are expressing only the change in  $w_{kj}$ .)

$$\Delta w_{kj} = \varepsilon \cdot \left[ \sum_{l=1}^q (y^{<1>} - O^{<1>}) \cdot O^{<1>} \cdot (1 - O^{<1>}) \cdot v_{lk} \right] \cdot [z_k \cdot (1 - z_k) \cdot x_j]$$

Equivalent notation with the saved variable  $\delta$ :

$$\Delta w_{kj} = \varepsilon \cdot \left( \sum_{l=1}^q \delta_l \cdot v_{lk} \right) \cdot [z_k \cdot (1 - z_k) \cdot x_j]$$

In **summary**, the equations for the backpropagation algorithm are:

Forward propagation of the input signal:

$$z_k = f \left( \sum_{j=0}^m w_{kj} \cdot x_j \right) = f(w_k \cdot x) \quad \dots \text{Output from the } k\text{th hidden neuron.}$$

$$O_l = f \left( \sum_{k=0}^p v_{lk} \cdot z_k \right) = f(v_l \cdot z) \quad \dots \text{Output from the } l\text{th output neuron.}$$

Backward propagation of the error to update the weights.

$$\delta_l = (y^{<l>} - O^{<l>}) \cdot O^{<l>} \cdot (1 - O^{<l>})$$

$$\Delta v_{lk} = \varepsilon \cdot \delta_l \cdot z_k$$

$$\Delta w_{kj} = \varepsilon \cdot \left( \sum_{l=1}^q \delta_l \cdot v_{lk} \right) \cdot [z_k \cdot (1 - z_k) \cdot x_j]$$

The first step calculates the discrepancy between the neural net model  $O$  and the actual target output  $y$ . The second step updates the weights between the hidden neurons and the output neurons.

Finally, we calculated the weights between the input neurons and the hidden neurons based on the weights calculated in the second step. We demonstrate the Mathcad implementation of the backpropagation algorithm with an example below.

Example: A Straight line with a neural network containing sigmoid functions  $f(\sigma) := \frac{1}{1 + e^{-\sigma}}$

Number of inputs:  $m := 1$   $j := 0..m$

Number of training patterns:  $N := 100$   $i := 0..N$

Dummy input (in lieu of threshold):  $x_{i,0} := 1$

One-dimensional input variable:  $x_{i,1} := \text{rnd}(2) - 1$

One-dimensional output variable:  $y_i := x_{i,1}$

Because the sigmoid function ranges [0, 1], we scale y to from [-1, 1] to [0.1, 0.9]

$\text{range}_{\text{new}} := 0.9 - 0.1$   $\text{range}_{\text{old}} := 2$   $\text{shift} := 1$

$$y_i := 0.1 + \frac{\text{range}_{\text{new}}}{\text{range}_{\text{old}}} \cdot (y_i - \text{shift})$$

Number of hidden neurons:  $p := 2$   $k := 0..p$   $kk := 1..p$

Assign initial weights randomly.

$v_k := \text{rnd}(2) - 1$

$w_{k,j} := \text{rnd}(2) - 1$

Present different samples and apply the backpropagation algorithm to calculate the weights.

Without programming, Mathcad has problem iterating sequentially in an efficient manner. We will put all variables into one

$$z_{i,kk} := f\left[\left(x \cdot w^T\right)_{i,kk}\right] \quad z_{i,0} := 1 \quad \varepsilon := 0.02$$

$$O_i := f\left[(z \cdot v)_{i,0}\right]$$

$$\delta_i := (y_i - O_i) \cdot O_i \cdot (1 - O_i)$$

$$v_k := v_k + \varepsilon \cdot \left[ \sum_{i=0}^N (\delta_i \cdot z_{i,k}) \right]$$

$$w_{k,j} := w_{k,j} + \varepsilon \cdot \sum_{i=0}^N \delta_i \cdot v_k \cdot z_{i,k} \cdot (1 - z_{i,k}) \cdot x_{i,j}$$