Sigmoidal Neurons

\[ f_{\text{net}}(t) = \frac{1}{1 + e^{-\text{net}(t)}} \]

This leads to a simplified form of the sigmoid function:

\[ S(\text{net}) = \frac{1}{1 + e^{-\text{net}}} \]

We do not need a modifiable threshold \( \theta \), because we will use “dummy” inputs as we did for perceptrons. The choice \( \tau = 1 \) works well in most situations and results in a very simple derivative of \( S(\text{net}) \).

In backpropagation networks, we typically choose \( \tau = 1 \) and \( \theta = 0 \). This leads to a simplified form of the sigmoid function: We do not need a modifiable threshold \( \theta \) because we will use “dummy” inputs as we did for perceptrons. The choice \( \tau = 1 \) works well in most situations and results in a very simple derivative of \( S(\text{net}) \).

\[ S(x) = \frac{1}{1 + e^{-x}} \]

\[ S'(x) = \frac{dS(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} \]

\[ \tau = 1 \]

\[ \tau = 0.1 \]

\[ \tau = 0.1 \]

\[ \tau = 0.1 \]

\[ \theta = 0.1 \]

\[ \theta = 0 \]

\[ \theta = 0 \]

\[ \theta = 0 \]

\[ \theta = 0 \]

\[ \theta = 0 \]

\[ \theta = 0 \]

Backpropagation Learning

Similar to the Adaline, the goal of the Backpropagation learning algorithm is to modify the network’s weights so that its output vector

\[ o_p = (o_{p,1}, o_{p,2}, \ldots, o_{p,K}) \]

is as close as possible to the desired output vector

\[ d_p = (d_{p,1}, d_{p,2}, \ldots, d_{p,K}) \]

for K output neurons and input patterns \( p = 1, \ldots, P \). The set of input-output pairs (exemplars) \( \{(x_p, d_p) | p = 1, \ldots, P\} \) constitutes the training set.

Backpropagation Learning

Also similar to the Adaline, we need a cumulative error function that is to be minimized:

Error = \( \sum_{p=1}^{P} \text{Err}(o_p, d_p) \)

We can choose the mean square error (MSE) once again (but the \( 1/P \) factor does not matter):

MSE = \( \frac{1}{P} \sum_{p=1}^{P} \sum_{j=1}^{K} (l_{p,j})^2 \)

where

\[ l_{p,j} = o_{p,j} - d_{p,j} \]

Terminology

Example: Network function \( f: R^3 \rightarrow R^2 \)

output vector \( \{o_1, o_2, o_3\} \)

output layer

hidden layer

input layer

input vector

\( W_{1,3}^{(1)} \)

\( W_{2,3}^{(2,1)} \)

\( W_{3}^{(1,0)} \)

\( W_{4,3}^{(1,0)} \)

\( W_{5,4}^{(2,1)} \)

\( W_{6,4}^{(2,1)} \)
For input pattern \( p \), the \( i \)-th input layer node holds \( x_{pi} \).

Net input to \( j \)-th node in hidden layer: \( net_{j}^{(1)} = \sum_{i=0}^{K} w_{ij}^{(0)} x_{pi} \)

Output of \( j \)-th node in hidden layer: \( x_{p,j}^{(1)} = S\left( \sum_{i=0}^{K} w_{ij}^{(0)} x_{pi} \right) \)

Net input to \( k \)-th node in output layer: \( net_{k}^{(2)} = \sum_{j=0}^{K} w_{kj}^{(2)} x_{p,j}^{(1)} \)

Output of \( k \)-th node in output layer: \( o_{p,k} = S\left( \sum_{j=0}^{K} w_{kj}^{(2)} x_{p,j}^{(1)} \right) \)

Network error for \( p \): \( E_p = \sum_{k=1}^{K} (t_{p,k} - o_{p,k})^2 \)

\( \partial E / \partial o_k \) can be disregarded as output units except \( o_k \):

\[
\frac{\partial E}{\partial o_k} = -2(d_k - o_k)
\]

Remember that \( o_k \) is obtained by applying the sigmoid function \( S \) to \( net_k^{(2)} \), which is computed by:

\[
net_k^{(2)} = \sum_{j=0}^{K} w_{kj}^{(2)} x_{p,j}^{(1)}
\]

Therefore, we need to apply the chain rule twice.

\[
\frac{\partial E}{\partial w_{kj}^{(2)}} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial x_{p,j}^{(1)}} \frac{\partial x_{p,j}^{(1)}}{\partial w_{kj}^{(2)}}
\]

We know that:

\[
\frac{\partial o_k}{\partial net_k^{(2)}} = S'(net_k^{(2)})
\]

Since \( net_k^{(2)} = \sum_{j=0}^{K} w_{kj}^{(2)} x_{p,j}^{(1)} \), we have:

\[
\frac{\partial net_k^{(2)}}{\partial w_{kj}^{(2)}} = x_{p,j}^{(1)}
\]

Which gives:

\[
\frac{\partial E}{\partial w_{kj}^{(2)}} = -2(d_k - o_k)S'(net_k^{(2)}) x_{p,j}^{(1)}
\]
Backpropagation Learning

As you surely remember from a few minutes ago:

\( S'(x) = S(x)(1-S(x)) \)

Then we can simplify the generalized error terms:

\[
\delta_i = (d_i - o_i)S'(net_i^{(1)})
\]

\[
\Rightarrow \delta_i = (d_i - o_i)o_i(1-o_i)
\]

And:

\[
\mu_j = \sum_{k=1}^{K} \delta_k w_{kj}^{(2,1)} S'(net_j^{(1)})
\]

\[
\Rightarrow \mu_j = \sum_{k=1}^{K} \delta_k w_{kj}^{(2,1)} x_j^{(1)}(1-x_j^{(1)})
\]

September 30, 2010 Neural Networks Lecture 8: Backpropagation Learning

Backpropagation Learning

Algorithm Backpropagation;

Start with randomly chosen weights;

while MSE is above desired threshold and computational bounds are not exceeded, do

for each input pattern \( x_p \), \( 1 \leq p \leq P \),

Compute hidden node inputs;

Compute hidden node outputs;

Compute inputs to the output nodes;

Compute the network outputs;

Compute the error between output and desired output;

Modify the weights between hidden and output nodes;

Modify the weights between input and hidden nodes;

end-for

end-while.