Multi-Layer Networks & Backpropagation
Multi-Layer Networks

- Generally much more versatile than single neurons

- No linear-separability requirement for problem space!

- Training approach is less obvious and potentially more time consuming.
Multi-Level Networks

- Several varieties, the most common of which is known as:
  - MLP (Multi-Level Perceptron)
  - Feed-forward network
  - Backpropagation Network (alluding to a common method of training these networks; other training methods could conceivably be used, so this is not a good name for the networks themselves.)
MLP Network

Note that sometimes the input is counted as a “layer”.
The real layers other than the output are called “hidden” layers.
Demo nnd11f

- Shows a simple 2-level network:
  - 1 input, 1 output
  - 2 neurons in first layer, with 1 weight and 1 bias each, logsig activation function
  - 1 neuron in output layer, with 2 weights and 1 bias
  - Output activation function selectable from: purelin (identity), tansig, logsig
- Plot is network output vs. input
Function Approximation Demo nnd11f

Input  Log-Sigmoid Layer  Linear Layer

\[ f^1(n) = \frac{1}{1 + e^{-n}} \]

\[ f^2(n) = n \]

Nominal Parameter Values

\[ w^1_{1,1} = 10 \quad w^1_{2,1} = 10 \quad b^1_1 = -10 \quad b^1_2 = 10 \]

\[ w^2_{1,1} = 1 \quad w^2_{1,2} = 1 \quad b^2 = 0 \]
Nominal Response
Parameter Variations

0 ≤ \( b_2^1 \leq 20 \)

-1 ≤ \( w_{1,1}^2 \) ≤ 1

-1 ≤ \( w_{2,2}^2 \) ≤ 1

-1 ≤ \( b_{1,2}^2 \) ≤ 1
Constructing a Multi-Layer Classifier

Design a network by hand that implements this decision problem.
Elementary Decision Boundaries

First Boundary:

\[ a_1^1 = \text{hardlim} \left( \begin{bmatrix} -1 & 0 \end{bmatrix} p + 0.5 \right) \]

Second Boundary:

\[ a_2^1 = \text{hardlim} \left( \begin{bmatrix} 0 & -1 \end{bmatrix} p + 0.75 \right) \]

First Subnetwork
Elementary Decision Boundaries

Third Boundary:

\[ a^1_3 = \text{hardlim}( \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{p} - 1.5 ) \]

Fourth Boundary:

\[ a^1_4 = \text{hardlim}( \begin{bmatrix} 0 & 1 \end{bmatrix} \mathbf{p} - 0.25 ) \]

Second Subnetwork
Total Network

\[
W^1 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad b^1 = \begin{bmatrix} 0.5 \\ 0.75 \\ -1.5 \\ -0.25 \end{bmatrix}
\]

\[
W^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad b^2 = \begin{bmatrix} -1.5 \\ -1.5 \end{bmatrix}
\]

\[
W^3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b^3 = \begin{bmatrix} -0.5 \end{bmatrix}
\]
How to Train a MLP?

- With a single neuron, it is not too hard to see how to adjust the weights based upon the error values. We’ve already seen a couple of ways.

- With a multi-layer network, it is less obvious. For one thing, what is the “error” for the neurons in non-final layers? Without these, we don’t know how to adjust.

- This is called the “credit assignment” problem (maybe should be “blame assignment”).
Backpropagation

- Werbos, in his Harvard PhD thesis in 1974 found a method, but it was not widely disseminated.

- Rumelhart and McClelland, in 1985, discovered the method, presumably independently, and popularized it under the current name.

- In mathematics, such methods are in the category of “optimization”.
Backpropagation

- The technique is gradient descent, as explained for Adalines.

- However, the computation of the gradient is less clear.
Backpropagation Training Cycle

- **Forward propagation**: Derive the activation values (the inputs to the activation functions) at each neuron, and the final output.

- **Compute the error** in the output.

- **Backpropagate** the error through the network to get “sensitivities” at each neuron. (The gradient approximation is derivable from the sensitivities.)

- Use the sensitivities to **derive weight changes**.

- Apply the weight changes.
Backpropagation Training Cycle

- Backpropagate is mathematically a lot like forward propagate.
- Sensitivities are used instead of signal values.
- The sensitivities are the partial derivatives of the MSE with respect to the activation values.
- Basically both are iterated matrix multiplications and applications of the activation functions of the neurons.
How to Adjust Weights?

- With a single neuron, it is not too hard to see how to adjust the weights based upon the error values. We’ve already seen a couple of ways.

- With a multi-layer network, it is less obvious. For one thing, what is the “error” for the neurons in non-final layers? Without these, we don’t know how to adjust.

- This is called the “credit assignment” problem (maybe should be “blame assignment”).
Multi-Layer Network

Each box has a row-vector of weights and a bias.

Each layer has a matrix of weights and a column vector of biases.
Multi-Layer Network

- Given an input vector, can compute the outputs.
- Given a sample, can compute the errors in output.
- Knowing gradient, can adjust the weights.
- Big Question: How to compute the gradient?
Multi-Layer Network

- Recall that the gradient consists of components \( \frac{\partial J}{\partial w} \)
  where \( J \) is the mean-squared error and \( w \) is some weight (including bias) in the network.

- For the generalized Adaline, already derived:

  \[
  \frac{\partial J}{\partial w_i} = -2 \varepsilon x_i f'(n),
  \]

  where \( x_i \) is the input corresponding to weight \( w_i \), and \( v (\text{net}) \) is the weighted sum. This works as is for the multi-layer case at the output layer.
Inside one neuron

The equation for updating the weight $w_i$ is given by the chain rule:

$$\frac{\partial J}{\partial w_i} = \left(\frac{\partial J}{\partial n}\right) \left(\frac{\partial n}{\partial w_i}\right)$$

Using the chain rule again,

$$= \left(\frac{\partial (d-f(n))^2}{\partial v}\right) \left(\frac{\partial n}{\partial w_i}\right)$$

$$= -2\epsilon f'(n) x_i$$

$$= s \times x_i$$

where $s = \left(\frac{\partial J}{\partial n}\right)$ is called the sensitivity
Chain Rule Refresher

\[
\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw}
\]

Example

\[
f(n) = \cos(n) \quad n = e^{2w} \quad f(n(w)) = \cos(e^{2w})
\]

\[
\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw} = (-\sin(n))(2e^{2w}) = (-\sin(e^{2w}))(2e^{2w})
\]

Application to Gradient Calculation

\[
\frac{\partial J}{\partial w_{i,j}^m} = \frac{\partial J}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^m}
\]
Last Layer

\[ \text{output} = f(n) \]
\[ n = \text{net} \]
Utility of Sensitivity

\[
\frac{\partial J}{\partial w_i} = (\frac{\partial J}{\partial n}) (\frac{\partial n}{\partial w_i}) \\
= s \ x_i
\]

*anywhere* in the network, not just at the final layer.
\[ \frac{\partial J}{\partial w_i} = (\frac{\partial J}{\partial n}) (\frac{\partial n}{\partial w_i}) \]

= \( s \times x_i \) (s is “sensitivity”)
Computing of Sensitivity

Unlike at the output, it is less clear how to compute the sensitivity at an *arbitrary* neuron.

The key idea is to use an iterative approach that starts with the sensitivities at the last layer and works backward toward the first layer.
Backward Propagation of Sensitivity
Backward Propagation of Sensitivity

Express desired as a weighted sum of known:

\[ s = f'(n) \sum w_j s_j \]
Vector Form

Express desired as a weighted sum of known:

\[ \mathbf{s} = f'(n) \sum w_j s_j \]

Note: over-dot means derivative.

Vector Form for entire \( m^{th} \) layer:

\[ \mathbf{s}^m = \hat{\mathbf{F}}^m (\mathbf{n}^m) (\mathbf{W}^{m+1})^T \mathbf{s}^{m+1} \]
Correctness

Why should \( s = f'(n) \sum w_j s_j \)

\[
s = \frac{\partial J}{\partial n^m} = \sum \left( \frac{\partial n^{m+1}}{\partial n^m} \right) \left( \frac{\partial J}{\partial n^{m+1}} \right)
\]

from the *vector* form of the chain rule

\[
s^m = \frac{\partial J}{\partial n^m} = \left( \frac{\partial n^{m+1}}{\partial n^m} \right)^T \frac{\partial J}{\partial n^{m+1}}
\]

Vector Form for entire \( m^{\text{th}} \) layer:

\[
s^m = F'(n^m)(W^{m+1})^T s^{m+1}
\]
The $s$ subscripts here are not related to sensitivities.
Correctness, continued

\[ s = \sum \left( \frac{\partial n^{m+1}}{\partial n^m} \right) \left( \frac{\partial J}{\partial n^{m+1}} \right) \]

\[ = \left( \frac{\partial}{\partial n^m} \right) \sum w_j f(n_j^m) \quad s_j \quad \text{by def. of } n^{m+1} \]

\[ = \sum w_j f'(n^m) \quad s_j \]

\[ = f'(n^m) \sum w_j s_j \]
Vector Form

\[ s^m = \frac{\partial J}{\partial n^m} = \left( \frac{\partial n^{m+1}}{\partial n^m} \right)^T \frac{\partial J}{\partial n^{m+1}} = F^m(n^m)(W^{m+1})^T \frac{\partial J}{\partial n^{m+1}} \]

\[ s^m = F^m(n^m)(W^{m+1})^T \cdot s^{m+1} \]

\[ \frac{\partial n^{m+1}}{\partial n^m} = W^{m+1} F^m(n^m) \quad \dot{F}^m(n^m) = \begin{bmatrix} f^m(n_1^m) & 0 & \ldots & 0 \\ 0 & f^m(n_2^m) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & f^m(n_{s_m}^m) \end{bmatrix} \]
Fully-Subscripted Alternatives to the Vector Forms

\[ n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j} a_j^{m-1} + b_i^m \]

\[ \frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1} \quad \frac{\partial n_i^m}{\partial b_i^m} = 1 \]

Sensitivity

\[ s_i^m \equiv \frac{\partial J}{\partial n_i^m} \]

Gradient

\[ \frac{\partial J}{\partial w_{i,j}^m} = s_i^m a_j^{m-1} \quad \frac{\partial J}{\partial b_i^m} = s_i^m \]
Fully-Subscripted Alternatives to the Vector Forms

\[
\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial}{\partial n_j^m} \left( \sum_{l=1}^{S^m} w_{i, l}^m a_i^m + b_i^{m+1} \right) = w_{i, j}^m \frac{\partial a_j^m}{\partial n_j^m}
\]

\[
\frac{\partial n_i^{m+1}}{\partial n_j^m} = w_{i, j}^{m+1} \frac{\partial f_i^m (n_j^m)}{\partial n_j^m} = w_{i, j}^{m+1} f_{i, j}^m (n_j^m)
\]

\[
f_{i, j}^m (n_j^m) = \frac{\partial f_i^m (n_j^m)}{\partial n_j^m}
\]
Vector Form for Last Layer, M

\[ s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (t - a)^T (t - a)}{\partial n_i^M} = \frac{\partial}{\partial n_i^M} \sum_{j=1}^{S^M} (t_j - a_j)^2 = -2(t_i - a_i) \frac{\partial a_i}{\partial n_i^M} \]

\[ \frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial f^M(n_i^M)}{\partial n_i^M} = f^M(n_i^M) \]

\[ s_i^M = -2(t_i - a_i) f^M(n_i^M) \]

\[ \mathbf{s}^M = -2 \hat{F}^M(n^M)(t - a) \]
Backpropagation Training Cycle

- **Forward propagation**: Derive the activation values (the inputs to the activation functions) at each neuron, and the final output.

- **Compute the error** in the output.

- **Backpropagate** the error through the network to get “sensitivities” at each neuron. (The gradient approximation is derivable from the sensitivities.)

- Use the sensitivities to **derive weight changes**.

- Apply the weight changes.
Backpropagation (Sensitivities)

The sensitivities are computed by starting at the last layer, and then propagating backwards through the network to the first layer.

$$s^M \rightarrow s^{M-1} \rightarrow \ldots \rightarrow s^2 \rightarrow s^1$$

$$s^M = -2 F^M(n^M)(t - a)$$

$$s^m = F^m(n^m)(W^{m+1}T^{m+1})s^{m+1}$$

where $F^m(n^m)$ is the diagonal matrix of activation function derivative values.

$$F^m(n^m) = \begin{bmatrix}
  f^m(n_1^m) & 0 & \ldots & 0 \\
  0 & f^m(n_2^m) & \ldots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \ldots & f^m(n_{S^m}^m)
\end{bmatrix}$$
Weight and Bias Update

(Here we are using $\alpha$ instead of $\eta$ for the learning rate.)

\[ W^{m}(k+1) = W^{m}(k) - \alpha s^{m}(a^{m-1})^T \]

\[ b^{m}(k+1) = b^{m}(k) - \alpha s^{m} \]
Fully-Subscripted Version of Weight Update

\[ w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha s_i a_j^{m-1} \quad b_i^m(k+1) = b_i^m(k) - \alpha s_i \]

\[ W^m(k+1) = W^m(k) - \alpha s^m a^{m-1}T \quad b^m(k+1) = b^m(k) - \alpha s^m \]

\[ s^m = \frac{\partial J}{\partial n^m} = \begin{bmatrix} \frac{\partial J}{\partial n_1^m} \\ \frac{\partial J}{\partial n_2^m} \\ \vdots \\ \frac{\partial J}{\partial n_s^m} \end{bmatrix} \]
Summary

Forward Propagation

\[ a^0 = p \]

\[ a^{m+1} = f^{m+1}(W^{m+1}a^m) \quad m = 0, 2, \ldots, M - 1 \]

\[ a = a^M \]

Backpropagation

\[ s^M = -2\dot{F}^M(n^M)(t - a) \]

\[ s^m = \dot{F}^m(n^m)(W^{m+1})s^{m+1} \quad m = M - 1, \ldots, 2, 1 \]

Weight Update

\[ W^m(k + 1) = W^m(k) - \alpha s^m(a^{m-1})^T \]

\[ b^m(k + 1) = b^m(k) - \alpha s^m \]
Exercise

- Derive the backprop equations symbolically for a simple network.
- Then use the equations to train the network.
Label the Levels

0 1  M = 2
Label the Signal Vectors or Lines

\[ a^0 \rightarrow a^1 \rightarrow a^2 \]

Vectors, superscript = level
Label the Signal Vectors or Lines

Lines, superscript = level,
Label the Net (Activation) Values

\[ a_0^1 \]

\[ a_0^2 \]

\[ a_1^1 \]

\[ a_1^2 \]

\[ n_1^1 \]

\[ n_1^2 \]

\[ n_2^1 \]

\[ a_2^1 \]
Label the Weights and Biases
Write the forward equations for activations

\[ n^1_1 = w^1_{11} a^0_1 + w^1_{12} a^0_2 + b^1_1 \]
\[ a^1_1 = f(n^1_1) \]

\[ n^1_2 = w^1_{21} a^0_1 + w^1_{22} a^0_2 + b^1_2 \]
\[ a^1_2 = f(n^1_2) \]

\[ n^2_1 = w^2_{11} a^1_1 + w^2_{12} a^1_2 + b^2_1 \]
\[ a^2_1 = f(n^2_1) \]
Write the backward equations for sensitivities

\[ s_{11} = w_{211} s_{21} f'(n_{11}) \]

\[ s_{12} = w_{212} s_{21} f'(n_{12}) \]

\[ s_{21} = -2(d_{21} - a_{21}) f'(n_{21}) \]
Note

● The summations for the backpropagated sensitivities have only one term in this example, since the following layer has only one neuron.
Write the Equations for Weight and Bias Update

\[
\begin{align*}
\Delta w_{11} &= -\alpha s_1 a_0^1 \\
\Delta w_{12} &= -\alpha s_1 a_0^2 \\
\Delta b_1 &= -\alpha s_1 \\
\Delta w_{21} &= -\alpha s_2 a_0^1 \\
\Delta w_{22} &= -\alpha s_2 a_0^2 \\
\Delta b_2 &= -\alpha s_2
\end{align*}
\]
Exercises

- Use the derivation to train the network to realize xor, where the first layer activations are logsig and the second layer is a hardlim.

- Carry out the preceding type of derivation for a 2-3-2 network (2 inputs, 3 middle neurons, 2 outputs).
Note on Training vs. Use

- Discontinuous functions such as hardlim, hardlims, etc. don’t have derivatives.

- Therefore we train the network with continuous approximations to these functions, then replace them with the discontinuous versions during usage:
  
  usage  \hspace{2cm} train with
  hardlim  \hspace{2cm} logsig
  hardlims  \hspace{2cm} tansig
Numeric Example

1-2-1 Network

Input  Log-Sigmoid Layer  Linear Layer
Initial Conditions

\[ W^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} \quad b^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} \quad W^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} \quad b^2(0) = \begin{bmatrix} 0.48 \end{bmatrix} \]
Forward Propagation

\[ a^0 = p = 1 \]

\[ a^1 = f^1(W^1 a^0 + b^1) = \text{logsig} \left( \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} a^0 + \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} \right) = \text{logsig} \left( \begin{bmatrix} -0.75 \\ -0.54 \end{bmatrix} \right) \]

\[ a^1 = \begin{bmatrix} \frac{1}{1 + e^{-0.75}} \\ \frac{1}{1 + e^{-0.54}} \end{bmatrix} = \begin{bmatrix} 0.321 \\ 0.368 \end{bmatrix} \]

\[ a^2 = f^2(W^2 a^1 + b^2) = \text{purelin} \left( \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} a^1 + \begin{bmatrix} 0.321 \\ 0.368 \end{bmatrix} + \begin{bmatrix} 0.48 \end{bmatrix} \right) = \begin{bmatrix} 0.446 \end{bmatrix} \]

\[ e = t - a = \left\{ 1 + \sin \left( \frac{\pi}{4} p \right) \right\} - a^2 = \left\{ 1 + \sin \left( \frac{\pi}{4} 1 \right) \right\} - 0.446 = 1.261 \]
Transfer Function Derivatives

\[
f^1(n) = \frac{d}{dn}\left(\frac{1}{1 + e^{-n}}\right) = \frac{e^{-n}}{(1 + e^{-n})^2} = \left(1 - \frac{1}{1 + e^{-n}}\right)\left(\frac{1}{1 + e^{-n}}\right) = (1 - a^1)(a^1)
\]

\[
f^2(n) = \frac{d}{dn}f(n) = 1
\]
Backpropagation

\[ s^2 = -2F^2(n^2)(t - a) = -2\left[f^2(n^2)\right](1.261) = -2\left[1\right](1.261) = -2.522 \]

\[ s^1 = F^1(n^1)(W^2)^T s^2 = \begin{bmatrix} (1-a_1^1)(a_1^1) & 0 \\ 0 & (1-a_2^1)(a_2^1) \end{bmatrix} \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} \begin{bmatrix} -2.522 \end{bmatrix} \]

\[ s^1 = \begin{bmatrix} (1-0.321)(0.321) & 0 \\ 0 & (1-0.368)(0.368) \end{bmatrix} \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} \begin{bmatrix} -2.522 \end{bmatrix} \]

\[ s^1 = \begin{bmatrix} 0.218 & 0 \\ 0 & 0.233 \end{bmatrix} \begin{bmatrix} -0.227 \\ 0.429 \end{bmatrix} = \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \]
Weight Update

\[ \alpha = 0.1 \]

\[ W^2(1) = W^2(0) - \alpha s^2(a^1)^T = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \\ 0.321 & 0.368 \end{bmatrix} \]

\[ W^2(1) = \begin{bmatrix} 0.171 & -0.0772 \end{bmatrix} \]

\[ b^2(1) = b^2(0) - \alpha s^2 = \begin{bmatrix} 0.48 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} = \begin{bmatrix} 0.732 \end{bmatrix} \]

\[ W^1(1) = W^1(0) - \alpha s^1(a^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix} \]

\[ b^1(1) = b^1(0) - \alpha s^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix} \]