Recall Linear Threshold Units

- The basic model:
  \[ \text{Output} = f(\sum w_j x_j) \]
- Where \( f \) can be one of:
  \[ f = \text{sign()} \text{ Value in \{-1, 1\}} \]
  \[ f = \text{step()} \text{ Value in \{0,1\}} \]
  \[ \sigma(a) = \frac{1}{1+e^{-a}} \]
  \[ p(f = 1) = \sigma(\sum w_j x_j) \]

Linear Sigmoid Units

- Today we will work with units whose output is a real value in [0,1]
  \[ \text{Output} = \hat{y} = \sigma(\sum w_j x_j) \]
  \[ \sigma(a) = \frac{1}{1+e^{-a}} \]
- This conveniently satisfies
  \[ \sigma'(a) = \frac{-e^{-a}}{(1+e^{-a})^2} = \sigma(a)(1 - \sigma(a)) \]

Linear Sigmoid Units

- Consider an example \((x,y)\)
- And error function
  \[ \text{Err} = \frac{1}{2} [y - \hat{y}]^2 = \frac{1}{2} [y - \sigma(\sum_j w_j x_j)]^2 \]
- Applying gradient descent
  \[ w_k = w_k - \eta \frac{\partial \text{Err}}{\partial w_k} \]
- We get the update rule
  \[ w_k = w_k + \eta (y - \hat{y}) \hat{y}(1 - \hat{y}) x_k \]

Multi Layer Networks

- Must first develop convenient notation
- This is different from single unit notation
- But it simplifies the exposition of the algorithm that follows
Multi Layer Networks

Must first develop convenient notation

• Denote input as before by \( x_1, \ldots, x_n \)
• An internal node is identified by its index \( i \), and its output is \( x_i \)
• All internal nodes are
• And the final output is \( x_N \)
• The link from unit \( j \) to \( i \) has weight \( w_{j,i} \)
• The sum at unit \( i \) is \( s_i = \sum_j w_{j,i} x_j \)
• The output at \( i \) is \( x_i = \sigma(s_i) = \sigma(\sum w_{j,i} x_j) \)

Example

First step: compute \( s_i, x_i, \) and \( \sigma'_i = x_i(1 - x_i) \)

\[
\begin{align*}
\eta &= 0.1 \\
w_{56} &= w_{67} = 1 \\
w_{35} &= w_{46} = 0.6 \\
w_{31} &= w_{45} = w_{23} = w_{24} = 1
\end{align*}
\]

Input example: \((x_1, x_2) = (2, 3)\)

Desired output: \( L = 0 \)

As before we get an example \((x, y)\).

• \( x \) specifies the input units \( x_1, \ldots, x_n \)
• \( y \) is the intended output of \( x_N \)

Nothing is known about intention for middle layers (a.k.a. hidden units)

Apply same error function

And gradient descent

The error function

\[
\text{Err} = \frac{1}{2} |y - x_N|^2
\]

Gradient update:

\[
w_{j,i} = w_{j,i} - \eta \frac{\partial \text{Err}}{\partial w_{j,i}}
\]

How can we calculate the gradient for an arbitrary \( w_{j,i} \) (at middle or top layer)?
Multi Layer Networks

- The error function
  \[ \text{Err} = \frac{1}{2} [y - x_N]^2 \]

- Gradient update:
  \[ w_{j,i} = w_{j,i} - \eta \frac{\partial \text{Err}}{\partial w_{j,i}} \]

- Two basic observations:
  \[ \frac{\partial \text{Err}}{\partial w_{j,i}} = \frac{\partial \text{Err}}{\partial s_i} \frac{\partial s_i}{\partial w_{j,i}} \]
  \[ \frac{\partial s_i}{\partial w_{j,i}} = \frac{\partial}{\partial w_{j,i}} \left( \sum_j w_{j,i} x_j \right) = x_j \]

  Just a derivative of linear function

Backpropagation Algorithm

- A few more steps (on the board) yield the Backpropagation algorithm

- Start by initializing all \( w_{j,i} \) to small random values

Backpropagation Algorithm

- Algorithm on previous slide updates after each example
- This is known as "stochastic gradient descent" (similar to perceptron)
- The standard Backpropagation algorithm makes multiple iterations over training set: in each iteration it collects the gradients from all examples in the training set and only then makes an update.

Illustration of Backpropagation

\[ \eta = 0.1 \]
\[ w_{56} = w_{67} = 1 \]
\[ w_{35} = w_{36} = w_{45} = w_{46} = 0.6 \]
\[ w_{13} = w_{14} = w_{23} = w_{24} = 1 \]

Input example: \((x_1, x_2) = (2, 3)\)
Desired output: \( L = 0 \)
Backpropagation Example

First Step: compute \( s_i, x_i, \) and \( \sigma'_i = x_i(1 - x_i) \)

\[
\begin{align*}
    s_1 &= 1 \times 2 + 1 \times 3 = 5 \\
    x_3 &= \frac{1}{1 + e^{-5}} = 0.993 \\
    \sigma'_3 &= 0.007 \\
    s_4 &= 5 \\
    x_4 &= 0.993 \\
    \sigma'_4 &= 0.007 \\
    s_5 &= 0.6 \times 0.993 + 0.6 \times 0.993 = 1.192 \\
    x_5 &= 0.767 \\
    \sigma'_5 &= 0.179 \\
    s_6 &= 1.192 \\
    x_6 &= 0.767 \\
    \sigma'_6 &= 0.179 \\
    s_7 &= 1 \times 0.767 + 1 \times 0.767 = 1.534 \\
    x_7 &= \frac{1}{1 + e^{-1.534}} = 0.823 \\
    \sigma'_7 &= 0.146
\end{align*}
\]

Second Step: compute \( \Delta_i \)

\[
\begin{align*}
    \Delta_7 &= -\sigma'_7 \times (L - x_7) = -0.146 \times (0 - 0.823) = 0.120 \\
    \Delta_5 &= \sigma'_5 w_{57} \Delta_7 = 0.179 \times 1 \times 0.120 = 0.021 \\
    \Delta_6 &= \Delta_5 \\
    \Delta_3 &= \sigma'_3 \left[w_{35} \Delta_5 + w_{36} \Delta_6\right] = 0.000176 \\
    \Delta_4 &= \Delta_3
\end{align*}
\]

Third Step: update weights

\[
\begin{align*}
    w_{13} &= w_{13} - \eta x_1 \Delta_3 = 1 - 0.1 \times 2 \times 0.000176 = 0.9999648 \\
    \ldots \\
    w_{35} &= w_{35} - \eta x_3 \Delta_5 = 0.6 - 0.1 \times 0.993 \times 0.021 = 0.5579 \\
    \ldots \\
    w_{57} &= w_{57} - \eta x_5 \Delta_7 = 1 - 0.1 \times 0.767 \times 0.120 = 0.9908 \\
    \ldots
\end{align*}
\]

Multiple Output Nodes

- All outputs share the same hidden layer
- Network identifies useful representations that are useful for all outputs
- Exactly same algorithm applies
- Forward pass identical
- Backward pass: each output unit calculates Delta using \( \Delta_N \) formula

Multi Layer Networks

- Not easy to optimize; the error surface has a lot of local minima
- Solutions:
  - Momentum:
    \[
    w_{j,i} = w_{j,i} - \eta \frac{\partial \text{Err}}{\partial w_{j,i}} + \alpha \text{ previous update}
    \]
  - Use multiple restarts and pick one with lowest training set error
  - ... many more recent techniques
What does the hidden layer do?

- Example: self-encoders

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Hidden</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>0.25</td>
<td>0.004</td>
</tr>
<tr>
<td>010000</td>
<td>0.11</td>
<td>0.003</td>
</tr>
<tr>
<td>001000</td>
<td>0.57</td>
<td>0.002</td>
</tr>
<tr>
<td>000100</td>
<td>0.57</td>
<td>0.002</td>
</tr>
<tr>
<td>000010</td>
<td>0.55</td>
<td>0.002</td>
</tr>
<tr>
<td>000001</td>
<td>0.54</td>
<td>0.002</td>
</tr>
</tbody>
</table>

Multi Layer Networks

- How to pick network size (and shape)?
- Similar to model selection in other models
  - cross validation
  - Combine fit + penalty
- How many updates?
  - Overfitting with large number of updates
  - Can do with large network and moderate number of updates

Using validation set for stopping criterion per number of updates.
Multi Layer Networks

- Renewed interest in Deep Networks in last decade
- Several schemes for special network structure and special node functions
- Several schemes for training
- Combination of these ideas with BigData
- Yields
- Impressive improvements in performance in vision and other applications

Convolutional Networks

- Architecture inspired by vision system
- Alternating layers of grid based structures
- Each node calculates local function on patch from previous layer

Convolutional Networks

- Alternate layers of:
  - "convolution layer" applies filter to patch from previous layer; weights repeat in all nodes (i.e. same filter)
  - "Pooling layer" combines multiple filters of same block
- Followed by fully connected layers

Deep Networks

- Autoencoders: similar to 8-3-8 idea.
- Network fragments can be used to learn one level of internal representations in an unsupervised manner
- Restricted Boltzmann Machines (RBM): a probabilistic model with similar intuitive role
- Stacking these gives a deep network
- Further supervised training of entire model after this step

Deep Networks

- Active area of research
- Still not well understood
- Public interest due to empirical success

Multi Layer Neural Networks

- Complex representation of functions
- Can be trained with gradient based methods
- But training can be tricky
- Hidden layer "learning representation"
- Recent work on deep networks adds special architecture and/or training procedures