Lecture 7 - Review

• ANN attempt to model the passively parallel computational architecture of the brain.
• A network consists of many units that are interconnected. Each connection has a weight
• Learning is finding the best combination of weights
• Input units, hidden layer units, output units.
• Input units take the training/test set as input
• Output layer produces the category/class/decision.
• In Lecture 7 we focused on training one unit to make correct predictions.
The Perceptron - Thresholded

- Linear decision boundary

\[ o(x_1, \ldots, x_n) = \begin{cases} 
1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\
-1 & \text{otherwise.}
\end{cases} \]

Sometimes we’ll use simpler vector notation:

\[ o(\vec{x}) = \begin{cases} 
1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\
-1 & \text{otherwise.}
\end{cases} \]
Perceptron Training Rule

\[ w_i \leftarrow w_i + \Delta w_i \]

\[ \Delta w_i = \eta(t - o)x_i \]

- Learning the AND function, rate = 0.05

<table>
<thead>
<tr>
<th>w0</th>
<th>w1</th>
<th>w2</th>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>t</th>
<th>w0,x0</th>
<th>w1,x1</th>
<th>w2,x2</th>
<th>o</th>
<th>delta_w0</th>
<th>delta_w1</th>
<th>delta_w2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.2</td>
<td>0.3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.2</td>
<td>0.3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0.2</td>
<td>0.3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>0</td>
</tr>
<tr>
<td>-0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-0.1</td>
<td>0</td>
<td>0.3</td>
<td>1</td>
<td>-0.1</td>
<td>0</td>
<td>-0.1</td>
</tr>
<tr>
<td>-0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-0.2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-0.2</td>
<td>0.1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-0.2</td>
<td>0</td>
<td>0.2</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-0.2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

CSI - 635 Lecture 8
Linear Units – No Threshold

• Gradient Descent (Delta Rule) (update weights after looking at all training data)
  - For each linear unit weight $w_i$, Do
    \[ \Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i \]
  - For each linear unit weight $w_i$, Do
    \[ w_i \leftarrow w_i + \Delta w_i \]

• Stochastic Gradient Descent (update weights after looking at each instance)
  - What important data issue does this entail?
Network of Neurons

Four Key Decisions To Make

• Arrange neurons in various layers.
• Deciding the type of connections among neurons for different layers, as well as among the neurons within a layer.
• Deciding the way a neuron receives input and produces output.
• Determining the strength of connection within the network
Layers and Connections?

• Layers
  – How many input nodes, hidden units, hidden layers, output units.
  – What happens if you have too many hidden units?

• Connections
  – Uni (Hierarchical) or bi-directional (resonance) between neurons
  – Connect to units in other layers or within a layer (re-current: form cliques)
  – Full or partial connections between layers
Types of Learning

• Unsupervised (Self Organizing Maps) Module B)
• Reinforcement
• Backpropagation
• Off-line vs On-line training
• Learning Rules
  – Hebb’s Rule (input and output neurons are active strengthen wait).
  – Hopfield Law (like Hebb’s rule but specifies magnitude)
  – Delta Rule (YAVOHR change weights to minimizes MSE)
  – Kohonen’s Learning Law neurons compete to learn.
• http://hem.hj.se/~de96klda/NeuralNetworks.htm#1.1 Method
Training a Network of Neurons

- Use the backpropagation algorithm
  - Gradient descent (can get stuck in local minima)
  - Error is summed over all outputs
  - Network of neurons allows complex decision boundaries. Input layer not neurons.
Hidden Layer and Latent Concepts - 1

A target function:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000000</td>
<td>100000000</td>
</tr>
<tr>
<td>010000000</td>
<td>010000000</td>
</tr>
<tr>
<td>001000000</td>
<td>001000000</td>
</tr>
<tr>
<td>000100000</td>
<td>000100000</td>
</tr>
<tr>
<td>000010000</td>
<td>000010000</td>
</tr>
<tr>
<td>000001000</td>
<td>000001000</td>
</tr>
<tr>
<td>000000100</td>
<td>000000100</td>
</tr>
<tr>
<td>000000010</td>
<td>000000010</td>
</tr>
<tr>
<td>000000001</td>
<td>000000001</td>
</tr>
</tbody>
</table>

Can this be learned??

CSI - 635 Lecture 8
Hidden Layer and Latent Concepts - 2

A network:

Learned hidden layer representation:

<table>
<thead>
<tr>
<th>Input</th>
<th>Hidden Values</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000000 → .89 .04 .08 → 100000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010000000 → .01 .11 .88 → 010000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001000000 → .01 .97 .27 → 001000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000100000 → .99 .97 .71 → 000100000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000010000 → .03 .05 .02 → 000010000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000001000 → .22 .99 .99 → 000001000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000100 → .80 .01 .98 → 000000100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001 → .60 .94 .01 → 00000001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What Type of Neuron to Use?

- Linear units? Perceptrons? Use sigmoid, tanh

σ(x) is the sigmoid function

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

Nice property: \( \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x)) \)

We can derive gradient decent rules to train

- One sigmoid unit

- Multilayer networks of sigmoid units → Backpropagation
Backpropagation Algorithm

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit $k$
     $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$
  3. For each hidden unit $h$
     $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{h,k} \delta_k$$
  4. Update each network weight $w_{i,j}$
     $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$
     where
     $$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$
## Backpropagation a Worked Example

### Table: Rates

<table>
<thead>
<tr>
<th>Rate</th>
<th>$w_{30}$</th>
<th>$w_{31}$</th>
<th>$w_{32}$</th>
<th>$w_{40}$</th>
<th>$w_{41}$</th>
<th>$w_{42}$</th>
<th>$w_{50}$</th>
<th>$w_{51}$</th>
<th>$w_{52}$</th>
<th>$\delta_5$</th>
<th>$\delta_3$</th>
<th>$\delta_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1000</td>
<td>-0.0500</td>
<td>0.0500</td>
<td>-0.0500</td>
<td>0.0500</td>
<td>-0.0500</td>
<td>0.0500</td>
<td>-0.0500</td>
<td>0.0500</td>
<td>-0.0500</td>
<td>-0.1217</td>
<td>-0.0015</td>
<td>0.0015</td>
</tr>
<tr>
<td>0.1000</td>
<td>-0.0502</td>
<td>0.0500</td>
<td>-0.0500</td>
<td>0.0502</td>
<td>-0.0500</td>
<td>0.0500</td>
<td>-0.0622</td>
<td>0.0441</td>
<td>-0.0562</td>
<td>0.1291</td>
<td>0.0014</td>
<td>-0.0018</td>
</tr>
<tr>
<td>0.1000</td>
<td>-0.0500</td>
<td>0.0501</td>
<td>-0.0500</td>
<td>0.0500</td>
<td>-0.0502</td>
<td>0.0500</td>
<td>-0.0557</td>
<td>0.0505</td>
<td>-0.0498</td>
<td>0.1285</td>
<td>0.0016</td>
<td>-0.0016</td>
</tr>
<tr>
<td>0.1000</td>
<td>-0.0498</td>
<td>0.0501</td>
<td>-0.0498</td>
<td>0.0498</td>
<td>-0.0502</td>
<td>0.0498</td>
<td>-0.0496</td>
<td>0.0566</td>
<td>-0.0430</td>
<td>-0.1222</td>
<td>-0.0017</td>
<td>0.0013</td>
</tr>
<tr>
<td>0.1000</td>
<td>-0.0500</td>
<td>0.0500</td>
<td>-0.0500</td>
<td>0.0499</td>
<td>-0.0501</td>
<td>0.0500</td>
<td>-0.0556</td>
<td>0.0507</td>
<td>-0.0493</td>
<td>-0.1214</td>
<td>-0.0015</td>
<td>0.0015</td>
</tr>
<tr>
<td>0.1000</td>
<td>-0.0502</td>
<td>0.0500</td>
<td>-0.0500</td>
<td>0.0501</td>
<td>-0.0501</td>
<td>0.0500</td>
<td>-0.0615</td>
<td>0.0447</td>
<td>-0.0555</td>
<td>0.1290</td>
<td>0.0014</td>
<td>-0.0018</td>
</tr>
<tr>
<td>0.1000</td>
<td>-0.0500</td>
<td>0.0501</td>
<td>-0.0500</td>
<td>0.0499</td>
<td>-0.0502</td>
<td>0.0500</td>
<td>-0.0550</td>
<td>0.0512</td>
<td>-0.0491</td>
<td>0.1284</td>
<td>0.0016</td>
<td>-0.0016</td>
</tr>
<tr>
<td>0.1000</td>
<td>-0.0499</td>
<td>0.0501</td>
<td>-0.0498</td>
<td>0.0498</td>
<td>-0.0502</td>
<td>0.0498</td>
<td>-0.0489</td>
<td>0.0573</td>
<td>-0.0423</td>
<td>-0.1223</td>
<td>-0.0018</td>
<td>0.0013</td>
</tr>
</tbody>
</table>

### Diagram

```
1 -- 2
|    |    |
    |    |
3 -- 4
   |    |
    |    |
5
```

### Table: I

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>t</th>
<th>net3</th>
<th>net4</th>
<th>o3</th>
<th>o4</th>
<th>o5</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.0500</td>
<td>0.0500</td>
<td>0.4875</td>
<td>0.5125</td>
<td>0.4872</td>
<td>0.2374</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-0.0002</td>
<td>0.0002</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.4830</td>
<td>0.2673</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-0.1000</td>
<td>0.1000</td>
<td>0.4751</td>
<td>0.5249</td>
<td>0.4856</td>
<td>0.2646</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-0.0495</td>
<td>0.0495</td>
<td>0.4877</td>
<td>0.5123</td>
<td>0.4890</td>
<td>0.2391</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.0500</td>
<td>0.0499</td>
<td>0.4875</td>
<td>0.5124</td>
<td>0.4860</td>
<td>0.2362</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-0.0002</td>
<td>0.0000</td>
<td>0.4999</td>
<td>0.5000</td>
<td>0.4833</td>
<td>0.2669</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-0.1000</td>
<td>0.0999</td>
<td>0.4751</td>
<td>0.5249</td>
<td>0.4859</td>
<td>0.2643</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-0.0496</td>
<td>0.0493</td>
<td>0.4876</td>
<td>0.5123</td>
<td>0.4894</td>
<td>0.2395</td>
</tr>
</tbody>
</table>

CSI - 635 Lecture 8
Error Gradient For Sigmoid Function

\[
\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2
\]

\[
= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2
\]

\[
= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)
\]

\[
= \sum_d (t_d - o_d) \left( -\frac{\partial o_d}{\partial w_i} \right)
\]

\[
= -\sum_d (t_d - o_d) \frac{\partial o_d}{\partial \text{net}_d} \frac{\partial \text{net}_d}{\partial w_i}
\]

But we know:

\[
\frac{\partial o_d}{\partial \text{net}_d} = \frac{\partial \sigma(\text{net}_d)}{\partial \text{net}_d} = o_d(1 - o_d)
\]

\[
\frac{\partial \text{net}_d}{\partial w_i} = \frac{\partial (\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d}
\]

So:

\[
\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (t_d - o_d)o_d(1 - o_d)x_{i,d}
\]
When Do We Stop Training

• Passing all of the data through the network is termed an epoch.

• How do we over-fit with a neural network?

• Stopping criteria
  – Fixed number of epochs
  – Training/validation set error is below some threshold
Insights into Backpropagation

- Gradient descent over entire network weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight momentum $\alpha$
  \[ \Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n - 1) \]
- Minimizes error over training examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations $\rightarrow$ slow!
- Using network after training is very fast
Reading for Next Classes

• Lecture 9 – 09/30/03 (104 – 117, 119 – 121)