Connectionist Models

Consider humans:

- Neuron switching time \( \sim .001 \) second
- Number of neurons \( \sim 10^{10} \)
- Connections per neuron \( \sim 10^{4-5} \)
- Scene recognition time \( \sim .1 \) second
- 100 inference steps doesn’t seem like enough
  \( \rightarrow \) much parallel computation

Properties of artificial neural nets (ANN’s):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically
When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

Examples:

- Speech phoneme recognition [Waibel]
- Image classification [Kanade, Baluja, Rowley]
- Financial prediction
ALVINN drives 70 mph on highways

**Perceptron**

\[ \sum_{i=0}^{n} w_i x_i \]

\[ o = \begin{cases} 
1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\
-1 & \text{otherwise} 
\end{cases} \]

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

Sometimes we’ll use simpler vector notation:

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

(a) $x_2$ $+$ $+$ $-$ $-$ $-$ $+$

(b) $x_2$ $+$ $+$ $-$ $+$

Represents some useful functions

- What weights represent $g(x_1, x_2) = AND(x_1, x_2)$?

But some functions not representable

- e.g., not linearly separable
- Therefore, we’ll want networks of these...
Perceptron training rule

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

where

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

Where:

- \( t = c(\vec{x}) \) is target value
- \( o \) is perceptron output
- \( \eta \) is small constant (e.g., .1) called learning rate
Perceptron training rule

Can prove it will converge

- If training data is linearly separable
- and $\eta$ sufficiently small
Gradient Descent

Gradient-Descent($training\_examples, \eta$)

Each training example is a pair of the form $\langle \bar{x}, t \rangle$, where $\bar{x}$ is the vector of input values, and $t$ is the target output value. $\eta$ is the learning rate (e.g., .05).

• Initialize each $w_i$ to some small random value

• Until the termination condition is met, Do
  – Initialize each $\Delta w_i$ to zero.
  – For each $\langle \bar{x}, t \rangle$ in $training\_examples$, Do
    * Input the instance $\bar{x}$ to the unit and compute the output $o$
    * 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
    $$w_i \leftarrow w_i + \Delta w_i$$
Summary

Perceptron training rule guaranteed to succeed if
- Training examples are linearly separable
- Sufficiently small learning rate $\eta$

Linear unit training rule uses gradient descent
- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate $\eta$
- Even when training data contains noise
- Even when training data not separable by $H$
Multilayer Networks of Sigmoid Units

head hid ... who'd hood

F1 F2

Sigmoid Unit

\[
\begin{align*}
\sigma(x) &= \frac{1}{1 + e^{-x}} \\
\text{Nice property: } \frac{d\sigma(x)}{dx} &= \sigma(x)(1 - \sigma(x))
\end{align*}
\]

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

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$
More on 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
A target function:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000 →</td>
<td>10000000</td>
</tr>
<tr>
<td>01000000 →</td>
<td>01000000</td>
</tr>
<tr>
<td>00100000 →</td>
<td>00100000</td>
</tr>
<tr>
<td>00010000 →</td>
<td>00010000</td>
</tr>
<tr>
<td>00001000 →</td>
<td>00001000</td>
</tr>
<tr>
<td>00000100 →</td>
<td>00000100</td>
</tr>
<tr>
<td>00000010 →</td>
<td>00000010</td>
</tr>
<tr>
<td>00000001 →</td>
<td>00000001</td>
</tr>
</tbody>
</table>

Can this be learned??
Learning Hidden Layer Representations

A network:

Learned hidden layer representation:

<table>
<thead>
<tr>
<th>Input</th>
<th>Hidden</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000000</td>
<td>.89 .04 .08</td>
<td>100000000</td>
</tr>
<tr>
<td>010000000</td>
<td>.01 .11 .88</td>
<td>010000000</td>
</tr>
<tr>
<td>001000000</td>
<td>.01 .97 .27</td>
<td>001000000</td>
</tr>
<tr>
<td>000100000</td>
<td>.99 .97 .71</td>
<td>000100000</td>
</tr>
<tr>
<td>000010000</td>
<td>.03 .05 .02</td>
<td>000010000</td>
</tr>
<tr>
<td>000001000</td>
<td>.22 .99 .99</td>
<td>000001000</td>
</tr>
<tr>
<td>000000100</td>
<td>.80 .01 .98</td>
<td>000000100</td>
</tr>
<tr>
<td>000000010</td>
<td>.60 .94 .01</td>
<td>00000001</td>
</tr>
</tbody>
</table>
Overfitting in ANNs
Neural Nets for Face Recognition

Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces
Learned Hidden Unit Weights

left  strt  rght  up

Learned Weights

Typical input images

http://www.cs.cmu.edu/~tom/faces.html
Recurrent Networks

(a) Feedforward network

(b) Recurrent network

(c) Recurrent network unfolded in time