• Most of this class has been focused on Symbolic AI
  • Focus or symbols and relationships between them
  • Search, logic, decision trees, etc.
• Assumption: Key requirement for intelligent behavior is the manipulation of symbols
• Neural networks are a little different: subsymbolic behavior
20-1: Biological Neurons
Biological Neurons

- Biological neurons transmit (more-or-less) electrical signals
- Each Nerve cell is connected to the “outputs” of several other neurons
- When there is a sufficient total input from all inputs, the cell “fires”, and sends outputs to other cells
- Extreme oversimplification, simple version is the model for Artificial Neural Networks
20-3: Artificial Neural Networks

-1

Bias weight

$w_0$

$w_1$

$w_2$

$w_3$

Inputs

Threshold Function or Activation Function

Output

Input weights

Σ
20-4: Activation Function

- Neurons are mostly binary
  - Fire, or don’t fire
  - Not “fire at 37%”
- Model this with an activation function
  - Step Function
  - Sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$
- Talk about why the sigmoid function can be better than the step function when we do training
A single Neuron can compute a simple function

What functions do each of these neurons compute?
A single Neuron can compute a simple function

What functions do each of these neurons compute?
Neural Networks

• Of course, things get more fun when we connect individual neurons together into a network
  • Outputs of some neurons feed into the inputs of other neurons
• Add special “input nodes”, used for input
20-8: Neural Networks

- Input Nodes
- Hidden Nodes
- Output Nodes
Neural Networks

- Feed Forward Networks
  - No cycles, signals flow in one direction
- Recurrent Networks
  - Cycles in signal propagation
  - Much more complicated: Need to deal with time, learning is much harder
- We will focus on Feed Forward networks
Feed Forward Neural Networks are *Nonlinear Function Approximators*

Output of the network is a function of its inputs

Activation function is non-linear, allows for representation of non-linear functions

Adjust weights, change function

Neural Networks are used to efficiently approximate complex functions
Common use for Neural Networks is classification

- We’ve already seen classification with decision trees and naive bayes
- Map inputs into one or more outputs
- Output range is split into discrete “classes” (like “spam” and “not spam”)

Useful for learning tasks where “what to look for” is unknown

- Face recognition
- Handwriting recognition
Perceptrons

- Feed Forward
- Single-layer network
- Each input is directly connected to one or more outputs
For each perceptron:

- Threshold firing function is used (not sigmoid)
- Output function $o$:
  - $o(x_1, \ldots x_n) = 1$ if $w_o + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0$
  - $o(x_1, \ldots x_n) = 0$ if $w_o + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \leq 0$
Since each perceptron is independent of others, we can examine each in isolation.

Output of a single perceptron:
- \[ \sum_{j=1}^{n} W_j x_j > 0 \]
- (or, \( W \cdot x > 0 \))

Perceptrons can represent any linearly separable function.

Perceptrons can only represent linearly separable functions.
20-16: Linearly Separable

Function: $X_1$ or $X_2$

Function: $X_1$ and $X_2$
20-17: Linearly Separable

Function: $X_1$ or $X_2$

Function: $X_1$ and $X_2$
Function: $X_1 \text{ xor } X_2$
Perceptron Learning

inputs : in1, in2, ..., inj
weights : w1, w2, ... wn
training examples: t1 = (tin1, to1),
                 t2 = (tin2, to2), ...

do
  for t in training examples
    inputs = tin
    o = compute output with current weights
    E = to - o
    for each w in weight
      wi = wi + alpha * tin[i] * E
while notConverged
If the output signal is too high, weights need to be reduced
- “Turn down” weights that contributed to output
- Weights with zero input are not affected

If output is too low, weights need to be increased
- “Turn up” weights that contribute to output
- Zero-input weights not affected

Doing a hill-climbing search through weight space
Learn the majority function with 3 inputs
  • (plus bias input)
• $\text{out} = 1$ if $\sum_j w_j in_j > 0$, 0 otherwise
• $\alpha = 0.2$
• Initially, all weights 0
20-22: **Perceptron Example**

<table>
<thead>
<tr>
<th>bias</th>
<th>inputs</th>
<th>expected out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 0 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1 0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0</td>
<td>0</td>
</tr>
</tbody>
</table>
20-23: Perceptron Example

<table>
<thead>
<tr>
<th>bias</th>
<th>inputs</th>
<th>expected out</th>
<th>w0</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>actual out</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 0 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 0 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 1 0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Perceptron Example

<table>
<thead>
<tr>
<th>bias</th>
<th>inputs</th>
<th>expected out</th>
<th>w0</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>actual out</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2 0.2 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 1 0 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2 0.2 0.2</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Perceptron Example

<table>
<thead>
<tr>
<th>bias</th>
<th>inputs</th>
<th>expected</th>
<th>w0</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>actual</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 1 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2 0 0.2 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0 0</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 0 1 1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 1 0 1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 1 0 0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Perceptron Example

<table>
<thead>
<tr>
<th>bias</th>
<th>inputs</th>
<th>expected out</th>
<th>w0</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>actual out</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2 0 0.2 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
</tbody>
</table>
## Perceptron Example

<table>
<thead>
<tr>
<th>bias</th>
<th>inputs</th>
<th>expected out</th>
<th>w0</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>actual out</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2 0 0.2 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0 0</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>-0.2 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 0 1 0</td>
<td>1</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 1 0 0</td>
<td>1</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>
### Perceptron Example

<table>
<thead>
<tr>
<th>bias</th>
<th>inputs</th>
<th>expected</th>
<th>w0</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>actual</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>-0.2 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0.2 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0 0.2</td>
</tr>
</tbody>
</table>
## Perceptron Example

<table>
<thead>
<tr>
<th>bias</th>
<th>inputs</th>
<th>expected</th>
<th>w0</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>actual</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>-0.2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Perceptron Example

<table>
<thead>
<tr>
<th>bias</th>
<th>inputs</th>
<th>expected out</th>
<th>w0</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>actual out</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2 0 0.2 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0 0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>-0.2 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 0 0 1</td>
<td>1</td>
<td>-0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0.2 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>1 1 0 0</td>
<td>1</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0 0.2 0 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0 0.2 0 0.2</td>
</tr>
</tbody>
</table>

Still hasn’t converged, need more iterations
After 3 more iterations (of all weights):

<table>
<thead>
<tr>
<th>bias</th>
<th>inputs</th>
<th>expected</th>
<th>w0</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>actual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>out</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>out</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0</td>
</tr>
</tbody>
</table>
What if we can’t learn the function exactly?
  - Function is not linearly separable
Want to do “as well as possible”
Minimize the sum of the squared error
\[ E = \sum (t_d - o_d)^2 \] for \( d \) in the training set
• Searching through a space of weights
• Much like local search, define an error $E$ as a function of the weights
• Find values of weights to minimize $E$
• Follow the gradient – largest negative change in $E$
  • Alas, $E$ is discontinuous, hard to differentiate
  • Instead of using actual output, use unthresholded output
Gradient Descent & Delta Rule

- Gradient descent: follow the steepest slope down the error surface
- Consider the derivative of $E$ with respect to each weight
- $E = \sum (t_d - o_d)^2$ for $d$ in the training set

\[
\frac{dE}{dw_i} = \sum 2(t_d - o_d) \frac{d(t_d - o_d)}{dw_i}
\]

First, we will simplify for looking at a single training data point (then we can sum over all of them, since the derivative of a sum is the sum of the derivatives)
For a single training example:

\[
\frac{dE}{dw_i} = \frac{d(t_d - o_d)^2}{dw_i} \\
= 2(t_d - o_d) \frac{d(t_d - o_d)}{dw_i}
\]

since

\[
\frac{d(f(x)^2)}{dx} = 2f(x) \frac{d(f(x))}{dx}
\]
For a single training example:

\[
\frac{dE}{dw_i} = 2(t_d - o_d) \frac{d(t_d - o_d)}{dw_i} \\
= 2(d_d - o_d) \frac{d(-x_idw_i)}{dw_i}
\]

- \(t_d\) doesn’t involve \(w_i\), so \(\frac{d(t_i)}{dw_i} = 0\)
- \(o_d = w_1x_{1d} + w_2x_{2d} + w_3x_{3d} + \ldots\), the only term that involves \(w_i\) is \(w_ix_{id}\)
For a single training example:

\[
\frac{dE}{dw_i} = 2(d_d - o_d) \frac{d(-x_idw_i)}{dw_i}
\]

\[
= 2(t_d - o_d)(-x_id)
\]

Since \( \frac{d(cx)}{dx} = c \)
Gradient Descent & Delta Rule

- Gradient descent: follow the steepest slope down the error surface
- Consider the derivative of $E$ with respect to each weight
- $E = \sum (t_d - o_d)^2$ for $d$ in the training set

$$\frac{dE}{dw_i} = \sum 2(t_d - o_d) \frac{d(t_d - o_d)}{dw_i}$$

$$= \sum 2(t_d - o_d)(-x_{id})$$

- Want to go down the gradient,
  $$\Delta w_i = \alpha \sum_{d \in D} (t_d - o_d)x_{id}$$
Gradient Descent & Delta Rule

- Gradient descent: follow the steepest slope down the error surface
- Consider the derivative of $E$ with respect to each weight
- After derivation, updating rule (called the Delta Rule) is:

$$\Delta w_i = \alpha \sum_{d \in D} (t_d - o_d) x_{id}$$

- $D$ is the training set, $\alpha$ is the training rate, $t_d$ is the expected output, and $o_d$ is the actual output, $x_{id}$ is the input along weight $w_i$. 
20-40: **Incremental Learning**

- Often not practical to compute global weight change for entire training set
- Instead, update weights incrementally
  - Observe one piece of data, then update
- Update rule: $w_i = \alpha(t - o)x_i$
  - Like perceptron learning rule – except uses unthresholded output
- Smaller training rate $\alpha$ typically used
- No theoretical guarantees of convergence
While perceptrons have the advantage of a simple learning algorithm, their computational limitations are a problem.

What if we add another “hidden” layer?

Computational power increases

- With one hidden layer, can represent any continuous function
- With two hidden layers, can represent any function

Example: Create a multi-layer network that computes XOR
While perceptrons have the advantage of a simple learning algorithm, their computational limitations are a problem.

What if we add another “hidden” layer?

Computational power increases
  - With one hidden layer, can represent any continuous function
  - With two hidden layers, can represent any function

Problem: How to find the correct weights for hidden nodes?
Multilayer Network Example

- Input units: $a_i$
- Hidden units: $a_j$
- Output units: $a_k$

Weights:
- $W_{ji}$
- $W_{kj}$
Backpropagation is an extension of the perceptron learning algorithm to deal with multiple layers of nodes. Our goal is to minimize the error function. To do this, we want to change the weights so as to reduce the error. We define error as a function of the weights like so:

- $E(w) = \text{expected} - \text{actual}$
- $E(w) = \text{expected} - g(\text{input})$

So, to determine how to change the weights, we compute the derivative of error with respect to the weights. This tells us the slope of the error curve at that
Nodes use sigmoid activation function, rather than the step function.

Sigmoid function works in much the same way, but is differentiable.

\[ g(input_i) = \frac{1}{1+e^{-input_i}}. \]

\[ g'(input_i) = g(input_i)(1 - g(input_i)) \] (good news here - calculating the derivative only requires knowing the output!)
Recall that our goal is to minimize the error function.

To do this, we want to change the weights so as to reduce the error.

We define error as a function of the weights like so:

\[ E(w) = \text{expected} - \text{actual} \]
\[ E(w) = \text{expected} - g(\text{input}) \]

So, to determine how to change the weights, we compute the derivative of error with respect to the weights.

This tells us the slope of the error curve at that point.
More on computing error

- \( E(w) = expected - g(input) \)
- \( E(w) = expected - g(w \ast i) \)
- \( \frac{dE}{dw} = 0 - g'(input)i \)
- \( \delta_w = \frac{dE}{dw} = -g(input) \ast (1 - g(input)) \ast i \)
Each weight is updated by $\alpha \times \Delta_i$

$W_{j,i} = W_{j,i} + \alpha \times a_j \times \Delta_i$
20-50: **Backpropagation**

- Updating input-hidden weights:
- Idea: each hidden node is responsible for a fraction of the error in $\delta_i$.
- Divide $\delta_i$ according to the strength of the connection between the hidden and output node.
- For each hidden node $j$
- $\delta_j = g(input)(1 - g(input)) \sum_{i \in \text{outputs}} W_{j,i} \delta_i$
- Update rule for input-hidden weights:
- $W_{k,j} = W_{k,j} + \alpha * input_k * \delta_j$
The whole algorithm can be summed up as:

While not done:
  for d in training set
    Apply inputs of d, propagate forward.
    for node $i$ in output layer
      $\delta_i = output \times (1 - output) \times (t_{exp} - output)\times \delta_i$
    for each hidden node $j$
      $\delta_j = output \times (1 - output) \times \sum W_{j,i} \delta_i$
  Adjust each weight
  $W_{j,i} = W_{j,i} + \alpha \times \delta_i \times input_j$
Theory vs Practice

- In the definition of backpropagation, a single update for all weights is computed for all data points at once.
  - Find the update that minimizes total sum of squared error.
- Guaranteed to converge in this case.
- Problem: This is often computationally space-intensive.
  - Requires creating a matrix with one row for each data point and inverting it.
- In practice, updates are done incrementally instead.
Stopping conditions

- Unfortunately, incremental updating is not guaranteed to converge.
- Also, convergence can take a long time.
- When to stop training?
  - Fixed number of iterations
  - Total error below a set threshold
  - Convergence - no change in weights
Backpropagation

- Also works for multiple hidden layers
- Backpropagation is only guaranteed to converge to a local minimum
  - May not find the absolute best set of weights
- Low initial weights can help with this
  - Makes the network act more linearly - fewer minima
- Can also use random restart - train multiple times with different initial weights.
Since backpropagation is a hillclimbing algorithm, it is susceptible to getting stuck in plateaus.

- Areas where local weight changes don’t produce an improvement in the error function.

A common extension to backpropagation is the addition of a momentum term.

- Carries the algorithm through minima and plateaus.

Idea: remember the “direction” you were going in, and by default keep going that way.

Mathematically, this means using the second derivative.
Momentum

- Implementing momentum typically means remembering what update was done in the previous iteration.

- Our update rule becomes:
  \[ \Delta w_{ji}(n) = \alpha \Delta_j x_{ji} + \beta \Delta w_{ji}(n-1) \]

- To consider the effect, imagine that our new delta is zero (we haven’t made any improvement)

- Momentum will keep the weights “moving” in the same direction.

- Also gradually increases step size in areas where gradient is unchanging.
  - This speeds up convergence, helps escape plateaus and local minima.
Design issues

- One difficulty with neural nets is determining how to *encode* your problem
  - Inputs must be 1 and 0, or else real-valued numbers.
  - Same for outputs
- Symbolic variables can be given binary encodings
- More complex concepts may require care to represent correctly.
Like some of the other algorithms we’ve studied, neural nets have a number of parameters that must be tuned to get good performance.

- Number of layers
- Number of hidden units
- Learning rate
- Initial weights
- Momentum term
- Training regimen

These may require trial and error to determine
Design issues

• The more hidden nodes you have, the more complex function you can approximate

• Is this always a good thing? That is, are more hidden nodes better?
• Overfitting
  • Consider a network with $i$ input nodes, $o$ output nodes, and $k$ hidden nodes
  • Training set has $k$ examples
  • Could end up learning a lookup table
20-61: **Overfitting**

Training Data

```
1101  100
0110  010
1111  001
0011  100
0000  010
```
20-62: **Overfitting**

**Training Data**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1101</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>001</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>010</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram showing a decision tree with data points and labels.](attachment:image.png)
20-63: Overfitting

Training Data

<table>
<thead>
<tr>
<th>1101</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td>010</td>
</tr>
<tr>
<td>1111</td>
<td>001</td>
</tr>
<tr>
<td>0011</td>
<td>100</td>
</tr>
<tr>
<td>0000</td>
<td>010</td>
</tr>
</tbody>
</table>
20-64: Overfitting

Training Data

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1101</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
20-65: Overfitting

Training Data
1101 100
0110 010
1111 001
0011 100
0000 010
20-66: Overfitting

Training Data
1101  100
0110  010
1111  001
0011  100
0000  010
20-67: Number Recognition

7 9 4
Number Recognition

Each pixel is an input unit
Number Recognition

[Diagram of a neural network with input features and output labels.]

- Fully Connected

Input features:
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
Recurrent NNs

- So far, we’ve talked only about feedforward networks.
  - Signals propagate in one direction
  - Output is immediately available
  - Well-understood training algorithms
- There has also been a great deal of work done on recurrent neural networks.
  - At least some of the outputs are connected back to the inputs.
This is a single-layer recurrent neural network
A Hopfield network has no special input or output nodes.

Every node receives an input and produces an output.

Every node connected to every other node.

Typically, threshold functions are used.

Network does not immediately produce an output.
  • Instead, it oscillates

Under some easy-to-achieve conditions, the network will eventually stabilize.

Weights are found using simulated annealing.
**Hopfield networks**

- Hopfield networks can be used to build an *associative memory*
- A portion of a pattern is presented to the network, and the net “recalls” the entire pattern.
- Useful for letter recognition
- Also for optimization problems
- Often used to model brain activity
Neural nets - summary

- Key idea: simple computational units are connected together using weights.
- Globally complex behavior emerges from their interaction.
- No direct symbol manipulation
- Straightforward training methods
- Useful when a machine that approximates a function is needed
  - No need to understand the learned hypothesis