Human Neurons

- **Switching time**
  - \(~ 0.001 \text{ second}\)
- **Number of neurons**
  - \(10^{10}\)
- **Connections per neuron**
  - \(10^{4-5}\)
- **Scene recognition time**
  - \(0.1 \text{ seconds}\)
- **Number of cycles per scene recognition?**
  - \(100 \rightarrow \text{much parallel computation!}\)
Perceptron as a Neural Network

This is one neuron:

- Input edges $x_1 \ldots x_n$, along with basis
- The sum is represented graphically
- Sum passed through an activation function $g$

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

\[ g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}} \]

Just change \( g \)!

- Why would we want to do this?
- Notice new output range \([0, 1]\). What was it before?
We train to minimize sum-squared error

\[ \ell(W) = \frac{1}{2} \sum_j [y^j - g(w_0 + \sum_i w_ix_i^j)]^2 \]

\[ \frac{\partial l}{\partial w_i} = - \sum_j [y_j - g(w_0 + \sum_i w_ix_i^j)] \frac{\partial}{\partial w_i} g(w_0 + \sum_i w_ix_i^j) \]

\[ \frac{\partial}{\partial w_i} g(w_0 + \sum_i w_ix_i^j) = x_i^j \frac{\partial}{\partial w_i} g(w_0 + \sum_i w_ix_i^j) = x_i^j g'(w_0 + \sum_i w_ix_i^j) \]

\[ \frac{\partial \ell(W)}{\partial w_i} = - \sum_j [y^j - g(w_0 + \sum_i w_ix_i^j)] x_i^j g'(w_0 + \sum_i w_ix_i^j) \]

Solution just depends on \( g' \): derivative of activation function!
Re-deriving the perceptron update

\[
\frac{\partial \ell(W)}{\partial w_i} = - \sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)] x_i^j g'(w_0 + \sum_i w_i x_i^j)
\]

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

\[
\frac{\partial \ell(W)}{\partial w_i} = - \sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)] x_i^j
\]

For a specific, incorrect example:

- \( w = w + y^*x \) (our familiar update!)
Sigmoid units: have to differentiate $g$

$$\frac{\partial \ell(W)}{\partial w_i} = - \sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)] x_i^j g'(w_0 + \sum_i w_i x_i^j)$$

$$g(x) = \frac{1}{1 + e^{-x}} \quad g'(x) = g(x)(1 - g(x))$$

$$w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j$$

$$\delta^j = [y^j - g(w_0 + \sum_i w_i x_i^j)] g^j(1 - g^j)$$

$$g^j = g(w_0 + \sum_i w_i x_i^j)$$
Aside: Comparison to logistic regression

- $P(Y|X)$ represented by:
  \[
P(Y = 1 \mid x, W) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}
  \]

- Learning rule – MLE:
  \[
  \frac{\partial \ell(W)}{\partial w_i} = \sum_j x_i^j [y^j - P(Y^j = 1 \mid x^j, W)]
  \]
  \[
  = \sum_j x_i^j [y^j - g(w_0 + \sum_i w_i x_i^j)]
  \]
  \[
  w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j
  \]
  \[
  \delta^j = y^j - g(w_0 + \sum_i w_i x_i^j)
  \]
Perceptron, linear classification, Boolean functions: $x_i \in \{0,1\}$

- Can learn $x_1 \lor x_2$?
  - $-0.5 + x_1 + x_2$
- Can learn $x_1 \land x_2$?
  - $-1.5 + x_1 + x_2$
- Can learn any conjunction or disjunction?
  - $0.5 + x_1 + \ldots + x_n$
  - $(-n+0.5) + x_1 + \ldots + x_n$
- Can learn majority?
  - $(-0.5*n) + x_1 + \ldots + x_n$
- What are we missing? The dreaded XOR!, etc.
Going beyond linear classification

Solving the XOR problem

\[ y = x_1 \text{ XOR } x_2 = (x_1 \land \neg x_2) \lor (x_2 \land \neg x_1) \]

\[ v_1 = (x_1 \land \neg x_2) \]
\[ = -1.5 + 2x_1 - x_2 \]

\[ v_2 = (x_2 \land \neg x_1) \]
\[ = -1.5 + 2x_2 - x_1 \]

\[ y = v_1 \lor v_2 \]
\[ = -0.5 + v_1 + v_2 \]
Hidden layer

- **Single unit:**

\[ \text{out}(x) = g(w_0 + \sum_i w_i x_i) \]

- **1-hidden layer:**

\[ \text{out}(x) = g \left( w_0 + \sum_k w_k g(w_0 + \sum_i w_i^k x_i) \right) \]

- **No longer convex function!**
Example data for NN with hidden layer

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??
A network:

Learned weights for hidden layer

Learned hidden layer representation:

<table>
<thead>
<tr>
<th>Input</th>
<th>Hidden</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Values</td>
<td></td>
</tr>
<tr>
<td>10000000</td>
<td>.89  .04  .08</td>
<td>100000000</td>
</tr>
<tr>
<td>01000000</td>
<td>.01  .11  .88</td>
<td>010000000</td>
</tr>
<tr>
<td>00100000</td>
<td>.01  .97  .27</td>
<td>001000000</td>
</tr>
<tr>
<td>00010000</td>
<td>.99  .97  .71</td>
<td>000100000</td>
</tr>
<tr>
<td>00001000</td>
<td>.03  .05  .02</td>
<td>000010000</td>
</tr>
<tr>
<td>00000100</td>
<td>.22  .99  .99</td>
<td>000001000</td>
</tr>
<tr>
<td>00000010</td>
<td>.80  .01  .98</td>
<td>000000100</td>
</tr>
<tr>
<td>00000001</td>
<td>.60  .94  .01</td>
<td>000000001</td>
</tr>
</tbody>
</table>
Learning the weights

Sum of squared errors for each output unit
Learning an encoding

Hidden unit encoding for input 01000000
NN for images

90% accurate learning head pose, and recognizing 1-of-20 faces
Weights in NN for images

Learned Weights

Typical input images
Forward propagation

1-hidden layer:

\[ \text{out}(x) = g \left( w_0 + \sum_k w_k g \left( w_0^k + \sum_i w_i^k x_i \right) \right) \]

Compute values left to right

1. Inputs: \( x_1, \ldots, x_n \)
2. Hidden: \( v_1, \ldots, v_n \)
3. Output: \( y \)
Back-propagation – learning

• Just gradient descent!!!
• Recursive algorithm for computing gradient
• For each example
  – Perform forward propagation
  – Start from output layer
    • Compute gradient of node $V_k$ with parents $U_1, U_2, ...$
    • Update weight $w_i^k$
    • Repeat (move to preceding layer)
Gradient descent for 1-hidden layer

\[ \ell(W) = \frac{1}{2} \sum_j [y^j - \text{out}(x^j)]^2 \]

\[ \text{out}(x) = g \left( \sum_{k'} w_{k'} g \left( \sum_{i'} w_{i' j} x_{i'} \right) \right) \]

\[ \frac{\partial \ell(W)}{\partial w_k} = \sum_{j=1}^m -[y^j - \text{out}(x^j)] \frac{\partial \text{out}(x^j)}{\partial w_k} \]

\[ \text{out}(x) = g \left( \sum_{k'} w_{k'} \text{v}_k^j \right) \quad \frac{\partial \text{out}(x)}{\partial w_k} = \text{v}_k^j g' \left( \sum_{k'} w_{k'} \text{v}_k^j \right) \]

Gradient for last layer same as the single node case, but with hidden nodes \( \text{v} \) as input!
Gradient descent for 1-hidden layer

\[ \ell(W) = \frac{1}{2} \sum_j [y^j - \text{out}(x^j)]^2 \]

\[ \text{out}(x) = g \left( \sum_{k'} w_{k'} g \left( \sum_{i'} w_{i'}^{k'} x_{i'} \right) \right) \]

\[ \frac{\partial \ell(W)}{\partial w^k_i} = \sum_{j=1}^m -[y - \text{out}(x^j)] \frac{\partial \text{out}(x^j)}{\partial w^k_i} \]

\[ \frac{\partial \text{out}(x)}{\partial w^k_i} = g' \left( \sum_{k'} w_{k'} g \left( \sum_{i'} w_{i'}^{k'} x_{i'} \right) \right) \frac{\partial}{\partial w^k_i} g \left( \sum_{i'} w_{i'}^{k'} x_{i'} \right) \]

Dropped \( w_0 \) to make derivation simpler

\[ \frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x) \]

For hidden layer, two parts:

- Normal update for single neuron
- Recursive computation of gradient on output layer
Forward propagation – prediction

• Recursive algorithm
• Start from input layer
• Output of node $V_k$ with parents $U_1, U_2, \ldots$:

$$V_k = g \left( \sum_i w_{i}^{k} U_i \right)$$
Back-propagation – pseudocode

Initialize all weights to small random numbers

• Until convergence, do:
  – For each training example $x,y$:
    1. Forward propagation, compute node values $V_k$
    2. For each output unit $o$ (with labeled output $y$):
       $$\delta_o = V_o(1-V_o)(y-V_o)$$
    3. For each hidden unit $h$:
       $$\delta_h = V_h(1-V_h) \sum_{k \text{ in output(h)}} w_{h,k} \delta_k$$
    4. Update each network weight $w_{i,j}$ from node $i$ to node $j$
       $$w_{i,j} = w_{i,j} + \eta \delta_j x_{i,j}$$
Multilayer neural networks

Inference and Learning:
• Forward pass: left to right, each hidden layer in turn
• Gradient computation: right to left, propagating gradient for each node
Convergence of backprop

• Perceptron leads to convex optimization
  – Gradient descent reaches \textit{global minima}

• Multilayer neural nets \textbf{not convex}
  – Gradient descent gets stuck in local minima
  – Selecting number of hidden units and layers = fuzzy process
  – NNs have made a HUGE comeback in the last few years!!!
    • Neural nets are back with a new name!!!!
      – Deep belief networks
      – Huge error reduction when trained with lots of data on GPUs
Overfitting in NNs

- Are NNs likely to overfit?
  - Yes, they can represent arbitrary functions!!!

- Avoiding overfitting?
  - More training data
  - Fewer hidden nodes / better topology
  - Regularization
  - Early stopping
Object Recognition

Slides from Jeff Dean at Google
Number Detection

Slides from Jeff Dean at Google
Acoustic Modeling for Speech Recognition

Close collaboration with Google Speech team

Trained in <5 days on cluster of 800 machines

30% reduction in Word Error Rate for English
(“biggest single improvement in 20 years of speech research”)

Launched in 2012 at time of Jellybean release of Android
2012-era Convolutional Model for Object Recognition

- Softmax to predict object class
- Fully-connected layers
- Convolutional layers (same weights used at all spatial locations in layer)
- Convolutional networks developed by Yann LeCun (NYU)

Basic architecture developed by Krizhevsky, Sutskever & Hinton (all now at Google).

Won 2012 ImageNet challenge with 16.4% top-5 error rate

Slides from Jeff Dean at Google
2014-era Model for Object Recognition

Module with 6 separate convolutional layers

24 layers deep!

Developed by team of Google Researchers:
Won 2014 ImageNet challenge with 6.66% top-5 error rate

Slides from Jeff Dean at Google
Good Fine-grained Classification

“hibiscus”

“dahlia”

Slides from Jeff Dean at Google
Good Generalization

Both recognized as a “meal”
Sensible Errors

“snake”

“dog”

Slides from Jeff Dean at Google
Works in practice for real users.

Wow.

The new Google plus photo search is a bit insane.

I didn’t tag those... :)

Slides from Jeff Dean at Google
Works in practice for real users.
What you need to know about neural networks

• Perceptron:
  – Relationship to general neurons

• Multilayer neural nets
  – Representation
  – Derivation of backprop
  – Learning rule

• Overfitting