Neural Networks

Le Song

Machine Learning I
CSE 6740, Fall 2013
Learning nonlinear decision boundary

- Linearly separable

- Nonlinearly separable

The XOR gate

Speech recognition
A decision tree for Tax Fraud

- **Input:** a vector of attributes
  - \( X = [\text{Refund}, \text{MarSt}, \text{TaxInc}] \)

- **Output:**
  - \( Y = \text{Cheating or Not} \)

- **H as a procedure:**

  - Each internal node: test one attribute \( X_i \)
  - Each branch from a node: selects one value for \( X_i \)
  - Each leaf node: predict \( Y \)
Apply model to test data I

Start from the root of tree.

Refund

Yes

No

NO

MarSt

Single, Divorced

Married

TaxInc

< 80K

> 80K

NO

YES

Query Data

<table>
<thead>
<tr>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>
Apply model to test data II

**Query Data**

<table>
<thead>
<tr>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

Decision Tree:
- **Refund**
  - Yes: NO
  - No: **MarSt**
    - Single, Divorced: **TaxInc**
      - < 80K: NO
      - > 80K: YES
    - Married: NO
Apply model to test data III

Query Data

<table>
<thead>
<tr>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

Refund
- Yes: NO
- No: MarSt
  - Single, Divorced: TaxInc
    - < 80K: NO
    - > 80K: YES
  - Married: NO
Apply model to test data IV

<table>
<thead>
<tr>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

**Query Data**

- Refund: No
- Marital Status: Married
- Taxable Income: 80K
- Cheat: ?

**Decision Tree:***

- **Refund**
  - Yes: NO
  - No:
    - **MarSt**
      - Single, Divorced
      - TaxInc:
        - < 80K: NO
        - > 80K: YES
      - Married: NO
Apply model to test data V

Query Data

<table>
<thead>
<tr>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

Assign Cheat to “No”
Expressiveness of decision tree

- Decision trees can express any function of the input attributes.
  - E.g., for Boolean functions, truth table row $\rightarrow$ path to leaf:

  Trivially, there is a consistent decision tree for any training set with one path to leaf for each example.

- Prefer to find more compact decision trees
Decision tree learning

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attrb1</th>
<th>Attrb2</th>
<th>Attrb3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Large</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Medium</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Small</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Medium</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Large</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Medium</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Large</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Small</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Medium</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Small</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attrb1</th>
<th>Attrb2</th>
<th>Attrb3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>No</td>
<td>Small</td>
<td>55K</td>
<td>?</td>
</tr>
<tr>
<td>12</td>
<td>Yes</td>
<td>Medium</td>
<td>80K</td>
<td>?</td>
</tr>
<tr>
<td>13</td>
<td>Yes</td>
<td>Large</td>
<td>110K</td>
<td>?</td>
</tr>
<tr>
<td>14</td>
<td>No</td>
<td>Small</td>
<td>95K</td>
<td>?</td>
</tr>
<tr>
<td>15</td>
<td>No</td>
<td>Large</td>
<td>67K</td>
<td>?</td>
</tr>
</tbody>
</table>

Training Set

Decision Tree

Test Set
Top-Down Induction of Decision tree

Main loop:
- $A \leftarrow$ the “best” decision attribute for next node
- Assign $A$ as the decision attribute for node
- For each value of $A$, create new descendant of node
- Sort training examples to leaf nodes
- If training examples perfectly classified, then STOP;
- ELSE iterate over new leaf nodes
Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.

Issues

- Determine how to split the records
  - How to specify the attribute test condition?
  - How to determine the best split?

- Determine when to stop splitting
Example of a decision tree

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Single</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Married</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Single</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Married</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Divorced</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Married</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Divorced</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Single</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Married</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Splitting Attributes

- **Refund**: Yes/No
- **Marital Status**: Single, Divorced, Married
- **Taxable Income**: < 80K, > 80K

**Model: Decision Tree**

**Training Data**
Another example of a decision tree

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Single</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Married</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Single</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Married</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Divorced</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Married</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Divorced</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Single</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Married</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

There could be more than one tree that fits the same data!

Training Data
How to determine the Best Split

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"

- Greedy approach:
  - Splitting nodes with **homogeneous** class distribution are preferred

- Need a measure of node impurity
How to compare attribute?

- **Entropy**
  - Entropy $H(X)$ of a random variable $X$

$$H(X) = - \sum_{i=1}^{N} P(x = i) \log_2 P(x = i)$$

- Entropy quantifies the randomness of a random variable.

- Information theory: $H(X)$ is the expected number of bits needed to encode a randomly drawn value of $X$ (under most efficient code)
Examples for computing Entropy

\[ H(X) = - \sum_{i=1}^{N} P(x = i) \log_2 P(x = i) \]

<table>
<thead>
<tr>
<th>C1</th>
<th>0</th>
<th>P(C1) = 0/6 = 0</th>
<th>P(C2) = 6/6 = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>6</td>
<td>Entropy = (-0 \log 0 - 1 \log 1 = -0 - 0 = 0)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C1</th>
<th>1</th>
<th>P(C1) = 1/6</th>
<th>P(C2) = 5/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>5</td>
<td>Entropy = (- (1/6) \log_2 (1/6) - (5/6) \log_2 (1/6) = 0.65)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C1</th>
<th>2</th>
<th>P(C1) = 2/6</th>
<th>P(C2) = 4/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>4</td>
<td>Entropy = (- (2/6) \log_2 (2/6) - (4/6) \log_2 (4/6) = 0.92)</td>
<td></td>
</tr>
</tbody>
</table>
Sample Entropy

- $S$ is a sample of training examples
- $p_+$ is the proportion of positive examples in $S$
- $p_-$ is the proportion of negative examples in $S$
- Entropy measure the impurity of $S$

$$H(S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_-$$
How to compare attribute?

- **Conditional Entropy of variable** $X$ **given variable** $Y$
  - The entropy $H(X|Y = j)$ of $X$ given a specific value $Y = j$:
    \[
    H(X|y = j) = - \sum_{i=1}^{N} P(x = i|y = j) \log_2 P(x = i|y = j)
    \]

- **Conditional entropy** $H(X|Y)$ of $X$ given $Y$: average of $H(X|Y = j)$
    \[
    H(X|Y) = - \sum_{j}^{N} P(y = j) H(X|y = j) = \sum_{i}^{N} \sum_{j}^{N} P(x = i, y = j) \log_2 \frac{P(x = i, y = j)}{P(y = j)}
    \]

- **Mutual information** (aka **information gain**) of $X$ given $Y$:
    \[
    I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X, Y)
    \]
Information Gain

- Information gain (after split a node):

\[
GAIN_{split} = \text{Entropy}(p) - \left( \sum_{i=1}^{k} \frac{n_i}{n} \text{Entropy}(i) \right)
\]

- \( n \) samples in parent node \( p \) is split into \( k \) partitions; \( n_i \) is number of records in partition \( i \)

- Measures Reduction in Entropy achieved because of the split. Choose the split that achieves most reduction (maximizes GAIN)
Stopping Criteria for Tree Induction

- Stop expanding a node when all the records belong to the same class

- Stop expanding a node when all the records have similar attribute values

- Early termination
Biologically inspired learning model

- Consider humans:
  - Neuron switching time
    ~ 0.001 second
  - Number of neurons
    ~ $10^{10}$
  - Connections per neuron
    ~ $10^{4-5}$
  - Scene recognition time
    ~ 0.1 second
  - 100 inference steps doesn't seem like enough
    → much parallel computation
Perceptron

- From biological neuron to artificial neuron (perceptron)

- Artificial neural nets (ANN)
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed processes
Jargon Pseudo-Correspondence

- Independent variable = input variable
- Dependent variable = output variable
- Coefficients = “weights”

Logistic Regression Model (the sigmoid unit) is a perceptron

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Coefficients</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age 34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gender 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stage 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

“Probability of being alive” = 0.6

Independent variables: $x_1$, $x_2$, $x_3$

Coefficients: $a$, $b$, $c$

Dependent variable: $p$ Prediction
What is logistic regression model

- Assume that the posterior distribution \( p(y = 1|x) \) take a particular form

\[
p(y = 1|x, \theta) = \frac{1}{1 + \exp(-\theta^\top x)}
\]

- Logistic function (or sigmoid function) \( \sigma(u) = \frac{1}{1+\exp(-u)} \)

\[
\frac{\partial \sigma(u)}{\partial u} = \sigma(u)(1 - \sigma(u))
\]
Learning parameters in logistic regression

- Find $\theta$, such that the conditional likelihood of the labels is maximized

$$\max_{\theta} l(\theta) := \log \prod_{i=1}^{m} P(y^i | x^i, \theta)$$

- Good news: $l(\theta)$ is concave function of $\theta$, and there is a single global optimum.

- Bad new: no closed form solution (resort to numerical method)
The objective function $l(\theta)$

- logistic regression model

$$p(y = 1|x, \theta) = \frac{1}{1 + \exp(-\theta^T x)}$$

- Note that

$$p(y = 0|x, \theta) = 1 - \frac{1}{1 + \exp(-\theta^T x)} = \frac{\exp(-\theta^T x)}{1 + \exp(-\theta^T x)}$$

- Plug in

$$l(\theta) = \log \prod_{i=1}^{m} P(y^i|x^i, \theta)$$

$$= \sum_{i} (y^i - 1) \theta^T x^i - \log(1 + \exp(-\theta^T x^i))$$
The gradient of $l(\theta)$

$$l(\theta) := \log \prod_{i=1}^{m} P(y^i | x^i, \theta)$$

$$= \sum_i (y^i - 1) \theta^\top x^i - \log(1 + \exp(-\theta^\top x^i))$$

- Gradient

$$\frac{\partial l(\theta)}{\partial \theta} = \sum_i (y^i - 1) x^i + \frac{\exp(-\theta^\top x^i) x^i}{1 + \exp(-\theta^\top x)}$$

- Setting it to 0 does not lead to closed form solution
Gradient Ascent algorithm

- Initialize parameter $\theta^0$

- Do

\[
\theta^{t+1} \leftarrow \theta^t + \eta \sum_i (y^i - 1) x^i + \frac{\exp(-\theta^t x^i)x^i}{1 + \exp(-\theta^t x)}
\]

- While the $||\theta^{t+1} - \theta^t|| > \epsilon$
Learning with square loss

- Find $\theta$, such that the conditional probability of the labels are close to the actual labels (0 or 1)

$$\min_{\theta} l(\theta) := \sum_{i}^{n} (y^i - P(y = 1|x^i, \theta))^2 = \sum_{i}^{n} (y^i - \sigma(\theta^\top x^i))^2$$

- Not a convex objective function

- Use gradient decent to find a local optimum
The gradient of $l(\theta)$

$$l(\theta) = \sum_{i=1}^{n} (y^i - P(y = 1|x^i, \theta))^2 = \sum_{i=1}^{n} (y^i - \sigma(\theta^\top x^i))^2$$

- Let $u^i = \theta^\top x^i$
- \[ \frac{\partial \sigma(u)}{\partial u} = \sigma(u)(1 - \sigma(u)) \]

Gradient

$$\frac{\partial l(\theta)}{\partial \theta} = \sum_{i} 2(y^i - \sigma(u^i)) \sigma(u^i) (1 - \sigma(u^i)) x^i$$
Gradient Descent algorithm

- Initialize parameter $\theta^0$

- Do

\[ u^i : = \theta^t \top x^i \]

\[ \theta^{t+1} \leftarrow \theta^t - \eta \sum_i 2(y^i - \sigma(u^i)) \sigma(u^i) (1 - \sigma(u^i)) x^i \]

- While the $||\theta^{t+1} - \theta^t|| > \epsilon$
Decision boundary a perceptron

NAND

\[ f(x_1 w_1 + x_2 w_2) = y \]

\[ f(0w_1 + 0w_2) = 1 \]
\[ f(0w_1 + 1w_2) = 1 \]
\[ f(1w_1 + 0w_2) = 1 \]
\[ f(1w_1 + 1w_2) = 0 \]

\[ \theta = 0.5 \]

\[ f(a) = \begin{cases} 1, & \text{for } a > \theta \\ 0, & \text{for } a \leq \theta \end{cases} \]

some possible values for \( w_1 \) and \( w_2 \)

<table>
<thead>
<tr>
<th>( w_1 )</th>
<th>( w_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.20</td>
<td>0.35</td>
</tr>
<tr>
<td>0.20</td>
<td>0.40</td>
</tr>
<tr>
<td>0.25</td>
<td>0.30</td>
</tr>
<tr>
<td>0.40</td>
<td>0.20</td>
</tr>
</tbody>
</table>
Difficult problem for perceptron

NAND

\[ f(x_1w_1 + x_2w_2) = y \]

\[
\begin{align*}
  f(0w_1 + 0w_2) &= 0 \\
  f(0w_1 + 1w_2) &= 1 \\
  f(1w_1 + 0w_2) &= 1 \\
  f(1w_1 + 1w_2) &= 0
\end{align*}
\]

\[ f(a) = \begin{cases} 
  1, & \text{for } a > \theta \\
  0, & \text{for } a \leq \theta
\end{cases} \]

some possible values for \( w_1 \) and \( w_2 \)
Connecting perceptrons = neural networks

\[ f(a) = \begin{cases} 
1, & \text{for } a > \theta \\
0, & \text{for } a \leq \theta 
\end{cases} \]

\[ \theta = 0.5 \text{ for all units} \]

A possible set of values for \((w_1, w_2, w_3, w_4, w_5, w_6)\):
\((0.6, -0.6, -0.7, 0.8, 1, 1)\)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>Z (color)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Neural Network Model

**Inputs**

- **Age**: 34
- **Gender**: 2
- **Stage**: 4

**Weights**

- Age to Hidden Layer 1: 0.6
- Gender to Hidden Layer 1: 0.1
- Stage to Hidden Layer 1: 0.3
- Age to Hidden Layer 2: 0.2
- Gender to Hidden Layer 2: 0.3
- Stage to Hidden Layer 2: 0.7

**Hidden Layer 1**

- Weight to Hidden Layer 2: 0.4

**Hidden Layer 2**

- Weight to Output: 0.5

**Output**

- Probability of being Alive: 0.6

**Independent variables**

- Age
- Gender
- Stage

**Weights**

- Hidden Layer to Output

**Dependent variable**

- Prediction
“Combined logistic models”

Inputs

<table>
<thead>
<tr>
<th>Age</th>
<th>Gender</th>
<th>Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Independent variables

Weights

Hidden Layer

Output

“Probability of being Alive”

Dependent variable

Prediction

Output

0.6
“Combined logistic models”

**Inputs**

- Age: 34
- Gender: 2
- Stage: 4

**Independent variables**

**Weights**

**Hidden Layer**

**Output**

“Probability of beingAlive”

**Dependent variable**

**Prediction**

- 0.6
“Combined logistic models”

Inputs

- **Age**: 34
- **Gender**: 1
- **Stage**: 4

Weights

- Input 1: .6
- Input 2: .1
- Input 3: .3
- Input 4: .7

Hidden Layer

- hidden layer weights: .4, .5, .8

Output

- **“Probability of being Alive”**: 0.6

*Independent variables*  |  *Weights*  |  *Hidden Layer*  |  *Weights*  |  *Dependent variable*  |  *Prediction*
---|---|---|---|---|---
Age | 34 | .6 | .4 | .5 | .8 | 0.6 | “Probability of being Alive”
Weights

Independent variables

Age 34
Gender 2
Stage 4

Weights

Hidden Layer

Prediction

Dependent variable

“Probability of being Alive”

0.6

Not really, no target for hidden units...
Training hidden units

- Find $\theta, \alpha, \beta$, such that the value of the output unit are close to the actual labels (0 or 1)

\[
\min_{\theta, \alpha, \beta} l(\theta, \alpha, \beta) := \sum_{i=1}^{n} (y^i - \sigma(\theta^T z^i))^2
\]

where

\[
\begin{align*}
    z^i_1 &= \sigma(\alpha^T x^i) \\
    z^i_2 &= \sigma(\beta^T x^i)
\end{align*}
\]

- Not a convex objective function

- Use gradient decent to find a local optimum
The gradient of $l(\theta, \alpha, \beta)$

$$l(\theta, \alpha, \beta):= \sum_{i}^{n}(y^i - \sigma(\theta^T z^i))^2$$

where $z_1^i = \sigma(\alpha^T x^i), z_2^i = \sigma(\beta^T x^i)$

- Let $u^i = \theta^T z^i$

Gradient

$$\frac{\partial l(\theta, \alpha, \beta)}{\partial \theta} = \sum_{i} 2(y^i - \sigma(u^i)) \sigma(u^i) \left( 1 - \sigma(u^i) \right) z^i$$

- $z^i$ is computed using $\alpha$ and $\beta$ from previous iteration

$$z_1^i = \sigma(\alpha^T x^i), z_2^i = \sigma(\beta^T x^i)$$
Gradient for $\alpha$ and $\beta$

- Use chain rule of derivatives

- Note $z^i_1 = \sigma(\alpha^\top x^i), z^i_2 = \sigma(\beta^\top x^i)$

- Let $\nu^i = \alpha^\top x^i$

Gradient

\[
\frac{\partial l(\theta, \alpha, \beta)}{\partial \alpha} = \frac{\partial l(\theta, \alpha, \beta)}{\partial z^i_1} \frac{\partial z^i_1}{\partial \alpha}
\]

\[
= \sum_i 2(y^i - \sigma(u^i)) \sigma(u^i)(1 - \sigma(u^i)) \theta_1 \sigma(\nu^i)(1 - \sigma(\nu^i)) x^i
\]
Expressive Capabilities of ANNs

- **Boolean functions:**
  - Every Boolean function can be represented by network with single hidden layer
  - But might require exponential (in number of inputs) hidden units

- **Continuous functions:**
  - Every bounded continuous function can be approximated with arbitrary small error, by network with one hidden layer [Cybenko 1989; Hornik et al 1989]
  - Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].
Abdominal Pain Perceptron

Male 
Age: 20
Temp: 37
WBC: 10
Pain Intensity: 1
Pain Duration: 1

Adjustable weights

Appendicitis
Diverticulitis
Ulcer
Duodenal
Perforated
Pain
Non-specific
Cholecystitis
Obstruction
Small Bowel
Pancreatitis
Myocardial Infarction Network

Durations Pain, Intensity Pain, Elevation ECG: ST, Smoker, Age, Male

Myocardial Infarction "Probability" of MI
The "Driver" Network

ALVINN [Pomerleau 1993]
Application: ANN for Face Reco.

- The model
- The learned hidden unit weights

Typical input images

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