Neural Networks

Non-symbolic Machine Learning

Russell and Norvig, Chapter 18 (18.7)
Overview: aims

• know what a neuron / unit is
• understand single-layer neural networks
• understand multi-layer neural networks
• know what a learning error is
• understand how backpropagation learning optimises weights by gradient descent
• know what a learning rate is (and how it affects learning)
• understand why/how weights and internal activations (hidden unit outputs) in the network change during learning
• have ideas of how to present data to a network
• know of the design-train-test methodology
Overview: topics

- Units
- Network
- Decision boundaries
- Multi-layer networks
- Learning in neural networks
- Gradient Descent
- Backpropagation
- Online and Batch learning
- Building a neural network
Neural networks are …

1. Computational models, consisting of simple processing elements, for representing and learning functions and procedures from examples

2. Abstract mathematical descriptions of biological neural circuitry and functionality

3. Tools for modelling cognitive phenomena

4. Tools for statistical classification and regression applications

Learning text to speech: NETtalk

A neural network

/Rk/

THE CAT CHASED

Rosenberg & Sejnowski (1986).
Driverless cars

NavLab/ALVINN
CMU, Pomerleau (1993)

Google’s self-driving car (2011)

http://www.youtube.com/watch?v=J17Qgc4a8xY&list=PLClLEUbT7kpOx0OrEf9kZSjDfRbQK-8K
The brain as a computational engine

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation units</td>
<td>1 CPU, 3x10^8 transistors (Intel Core 2 Duo)</td>
<td>10^{11} neurons</td>
</tr>
<tr>
<td>Storage units</td>
<td>10^{10} bits RAM, 10^{12} bits disk</td>
<td>10^{11} neurons</td>
</tr>
<tr>
<td>Cycle time</td>
<td>3x10^{-10} seconds</td>
<td>10^{-3} seconds</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>10^{10} bits / second</td>
<td>10^{14} bits / second</td>
</tr>
</tbody>
</table>

Neurons and their signals are more complex than bits and logic gates, but how much more is arguable. Computers are faster, but simpler devices.
Early models of neurons

- This first artificial neuron was invented by Warren McCulloch and Walter Pitts in 1943.
- They were colleagues of Norbert Wiener (invented “Cybernetics”), John von Neumann (helped invent modern computer architecture) and Claude Shannon (information theory).
- Their neuron takes input values (on artificial dendrites), multiplies by a weight (synaptic strength), processes these weighted inputs in a cell, and produces a 1 or 0 signal along the output (artificial axon).
The neuron: from biology to computational model

Pyramidal cells

©HPC@EPFL
Components of an artificial neural network

Structure
• Units ("logic" devices, non-linearities)
• Links (connections, wires, transport routes, dendrites and axons)
• Weights on the links (also called synapses, these are the adjustable parameters of the model, $w_j$)

Dynamics
• Update rule for the units (activity, $a_i$)
• Update rule for the weights (learning rule for long term memory)

Function
• Input / Output (environment, $(x_t, y_t)$, paradigm eg classification, regression)
The unit

\[ y = f(\text{net}) \]

\[ x_0 = 1 \quad \text{bias } \beta \]

\[ w_0 \]

\[ x_1 \quad w_1 \]

\[ \ldots \]

\[ x_n \quad w_n \]

Input Links | Input Function | Activation Function | Output | Output Links
---|---|---|---|---
Unit: linear component

\[ \text{net} = \sum_{i=1}^{n} x_i w_i + \beta \]

\( n \) inputs \( x_i \), \( n \) weights \( w_i \), bias \( \beta \), output \( y \).

```
public double sum(double[] x, double[] w, double bias){
    double sum=bias;
    for (int i=1; i<x.length; i++)
        sum+=x[i]*w[i];
    return sum;
}
```

Matlab

\[ \text{sum} = \text{X.W}+\beta \]
Unit: non-linear component

Regression (continuous output)

\[ f(\text{net}) = \frac{1}{1 + e^{-\text{net}}} \]

public double outputFunction(double net) {
    // a sigmoid function
    return 1.0/(1.0+Math.exp(-net));
}

Classification (hard threshold)

Can be used for logic functions (AND, OR, NOT etc.

public double outputFunction(double net) {
    // threshold function
    return (net>=0.0?1.0:0.0);
}
### Boolean Logic functions

#### $Y = X_1 \text{ OR } X_2$

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$Y$</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>1</td>
</tr>
</tbody>
</table>

#### $Y = X_1 \text{ AND } X_2$

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### $Y = \text{not } X_1$

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### $Y = X_1 \text{ XOR } X_2$

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$Y$</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>
Logic functions: OR

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>net</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Task: find \( w_1, w_2, \) and \( \beta \) to implement OR

What happens if \( w_1 = w_2 = \beta = 1? \)

\[
y = \begin{cases} 
0 & \text{if } x_1 w_1 + x_2 w_2 + \beta < 0 \\
1 & \text{if } x_1 w_1 + x_2 w_2 + \beta \geq 0
\end{cases}
\]

What about \( \beta = 0? \)

\( \beta = -0.5 \)
Logic functions: OR

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>net</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-0.5</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1.5</td>
<td>1</td>
</tr>
</tbody>
</table>

Task: find $w_1$, $w_2$, and $\beta$ to implement OR

What happens if $w_1 = w_2 = \beta = 1$?

$y = \begin{cases} 0 & \text{if } x_1w_1 + x_2w_2 + \beta < 0 \\ 1 & \text{if } x_1w_1 + x_2w_2 + \beta \geq 0 \end{cases}$

What about $\beta = 0$?

$\beta = -0.5$
Logic functions: OR

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>net</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(\beta)</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>(w_2 + \beta)</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>(w_1 + \beta)</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>(w_1 + w_2 + \beta)</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
x_1w_1 + x_2w_2 + \beta = \text{net}
\]

1. \(w_1 \cdot 0 + w_2 \cdot 0 + \beta < 0\)  
   \(\beta < 0\)

2. \(w_1 \cdot 0 + w_2 \cdot 1 + \beta \geq 0\)  
   \(w_2 + \beta \geq 0\)
   \(w_2 \geq -\beta\)

3. \(w_1 \cdot 1 + w_2 \cdot 0 + \beta \geq 0\)  
   \(w_1 + \beta \geq 0\)
   \(w_1 \geq -\beta\)

4. \(w_1 \cdot 1 + w_2 \cdot 1 + \beta \geq 0\)  
   \(w_1 + w_2 + \beta \geq 0\)
   \(w_1 + w_2 \geq -\beta\)
Decision boundary: OR

\[\begin{align*}
    w_1 &= 1.0 \\
    w_2 &= 1.0 \\
    \beta &= -0.5 \\
\end{align*}\]

\[x_1 = -x_2 + 0.5\]
Logic functions: AND

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>net</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Task: find $w_1$, $w_2$, and $\beta$ to implement AND

Let $w_1 = w_2 = 1$

$$y = \begin{cases} 0 & \text{if } x_1w_1 + x_2w_2 + \beta < 0 \\ 1 & \text{if } x_1w_1 + x_2w_2 + \beta \geq 0 \end{cases}$$

$\beta = ?$

$net = x_1w_1 + x_2w_2 + \beta$

$f(net) = \begin{cases} 0 & \text{if } net < 0 \\ 1 & \text{if } net \geq 0 \end{cases}$
Decision boundary: AND

\[ w_1 = 1.0 \]
\[ w_2 = 1.0 \]
\[ \beta = ? \]

\[ x_1 = -x_2 + 1.5 \]
What happens if the weights are negative?

\[ w_1 = -1.0 \]
\[ w_2 = -1.0 \]
\[ \beta = 0.5 \]

Decision boundary: NOR (NOT-OR)
What a network “computes”?

- Weight values are coefficients in a function
- Consider a classification problem: which group does a new point belong to?

A network with two inputs: $x_1$ and $x_2$
and one output: 0 or 1

Assume bias $\beta=0$ for now.

\[
\text{net} = \sum_{i=1}^{n} x_i w_i + \beta
\]

\[
f(\text{net}) = \begin{cases} 
0 & \text{if } \text{net} < 0 \\
1 & \text{if } \text{net} \geq 0
\end{cases}
\]
Decision boundaries (1)

- Decision boundary separates those input values that produce a 1 from those that produce 0.
- It is the task of any classifier to partition the input space into classes via decision boundaries.
The decision boundary contains all the values of the vector $x$ for which this sum $= 0$.

For these weights, it is zero when $x_1 w_1 + x_2 w_2 = 0$.

Here, that's $x_1 - 2x_2 = 0$, i.e.: the line $x_2 = 0.5 x_1$ (red line).
Decision boundaries (3)

- For a threshold unit, the decision boundary contains all the values for which \( \text{net} = 0 \)
- If bias = 0, the decision boundary must pass through the origin
- The bias value frees the decision boundary from crossing the origin

\[
\text{net} = \sum_{i=1}^{n} x_i w_i + \beta \\
\text{net} = x_1 w_1 + x_2 w_2 + \beta \\
0 = x_1 w_1 + x_2 w_2 + \beta \\
x_1 w_1 = -x_2 w_2 - \beta \\
x_1 = -x_2 \frac{w_2}{w_1} - \frac{\beta}{w_1}
\]
Decision boundaries (4)

- What if the samples are distributed differently…?
Decision boundaries (5)

- The bias value frees the decision boundary from crossing the origin.
- Allows the threshold unit to realize any linear separator.

\[ y = x_1 w_1 + x_2 w_2 + \beta \]
Logic functions: AND (1)

\[ x_1 w_1 + x_2 w_2 + \beta = \text{net} \]

1. \( w_1 \cdot 0 + w_2 \cdot 0 + \beta < 0 \)
   \[ \beta < 0 \]

2. \( w_1 \cdot 0 + w_2 \cdot 1 + \beta < 0 \)
   \[ w_2 + \beta < 0 \]
   \[ w_2 < -\beta \]

3. \( w_1 \cdot 1 + w_2 \cdot 0 + \beta < 0 \)
   \[ w_1 + \beta < 0 \]
   \[ w_1 < -\beta \]

4. \( w_1 \cdot 1 + w_2 \cdot 1 + \beta \geq 0 \)
   \[ w_1 + w_2 + \beta \geq 0 \]
   \[ w_1 + w_2 \geq -\beta \]
Logic functions: AND (2)

\[ net = \sum_{i=1}^{n} x_i w_i + \beta \]

\[ = x_1 w_1 + x_2 w_2 + \beta \]

\[ f(net) = \begin{cases} 
0 & \text{if } net < 0 \\
1 & \text{if } net \geq 0 
\end{cases} \]

\[ w_1 = 1.0 \]
\[ w_2 = 1.0 \]
\[ \beta = -1.5 \]

\[ x_1 = -x_2 \frac{w_2}{w_1} - \frac{\beta}{w_1} \]
\[ x_1 = -x_2 + 1.5 \]

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>net</th>
<th>f(net)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-0.5</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-0.5</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
</tr>
</tbody>
</table>
Logic functions: **XOR**

$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$= x_1 w_1 + x_2 w_2 + \beta$$

$$f(net) = \begin{cases} 
0 & \text{if } net < 0 \\
1 & \text{if } net \geq 0 
\end{cases}$$

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>net</th>
<th>f(net)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>?</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
<td>0</td>
</tr>
</tbody>
</table>
Each unit is a single decision boundary, so XOR cannot be computed by a single unit.
Single layer networks can only represent linearly separable functions
Making a network

Here, really just $k$ separate neurons, each with one output, which each use the same set of inputs.

```java
double[][] weights; // all weights
double[] biases; // all biases

double[] network(double[] x) {
    double[] y = new double[weights.length];
    for (int k = 0; k < weights.length; k++) {
        y[k] = outputFunction(sum(x, weights[k], biases[k]));
    }
    return y;
}
```
Multi-layer network

- Activation flows from input to output through a set of intermediate hidden (or latent) nodes.
- The activations of the hidden nodes are decided internally by the network.
- They depend upon the weights between input and hidden nodes.
How does that work?

• What is really happening is that the inputs are being transformed via input-hidden weights into a new space.
• This space may have more or less dimensions.
• We are hoping to choose weights so that the problem to solve at the hidden-output layer is then linearly separable.
What needs to be chosen?

- Whether to have a hidden layer or not (generally more capable with one)
- How many hidden units to use?
- Values for each of the weights (and biases) to solve our problem of interest
- Note: can use more than one hidden layer, but not usually beneficial
How to make these choices?

- Choose to have a hidden layer – might as well if using neural networks
- Number of hidden units: can start at 1 and add 1 at a time, comparing by somehow estimating performance
- Or can start with a big hidden layer and prune down
- Can consider pruning inputs (variable selection) and individual weights also
Solving XOR (1)

\[ Y = X_1 \text{ XOR } X_2 \]

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>Y</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>
Solving XOR (2)

$Y = X_1 \text{ XOR } X_2$

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>Y</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>

- Assume each node implements the threshold function
- Using a hidden layer, find values for the weights and biases to implement XOR
- Can be solved with integer weights
Solving XOR (3)

Y = X1 XOR X2

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>H1</th>
<th>H2</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Y = X1 XOR X2

w_{13} = 1.0
w_{14} = 1.0
w_{23} = 1.0
w_{24} = 1.0
w_{35} = 1.0
w_{45} = 1.0

β_3 = -0.5
β_4 = -1.5
Solving XOR (4)

\[ Y = X_1 \text{ XOR } X_2 \]

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>H1</th>
<th>H2</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\beta_3 &= -0.5 \\
\beta_4 &= -1.5 \\
\beta_5 &= -0.5
\end{align*}
\]
Decision boundaries with a hidden layer

• In general, decision boundaries in $x$, can be any complex curves
Multiple solutions for XOR

- 1\textsuperscript{st} solution: nodes 3 and 5 just copy $x_1$ and $x_2$ through.
- Node 4 does $x_1 \text{ AND} x_2$.
- Result is 1 if $x_1 \text{ OR} x_2$, but not if $x_1 \text{ AND} x_2$.

- Node 3 does AND, node 4 does OR.
- Node 6 calculates OR – AND, and the threshold only allows XOR through.
Neural Network Structures

• **Feedforward**
  • Single layer
    • E.g. Perceptron

• Multi layer

• **Recurrent**
  • Any neural network with at least one feedback connection
Break
Learning in neural networks

• Single-layer networks

• Outputs:
  – Binary output: function = binary threshold
  – Continuous output: function = sigmoid

\[
f(\text{net}) = \frac{1}{1 + e^{-\text{net}}}
\]
Notation: counts and indices

• Textbook is not consistent: uses $n$ for both number of inputs and number of training patterns.

• Hereon in slides, will use:
  – Observations/patterns: $i=1,..,n$.
  – Training patterns: $i=1,..,n_{tr}$
  – Test patterns: $i=1,..,n_{te}$
    • If using all observations for training, $n_{tr} = n$. If test set is a subset of the observations, $n_{tr} + n_{te} = n$.
  – Inputs/attributes: $j=1,..,p$.
  – Outputs/responses: $k=1,..,m$
Notation: data and network output (1)

• Observations/patterns:
  – input: $x_i = \{x_{ij}\}$, $i =$ pattern #, $j =$ input #
  – correct output: $y_i = \{y_{ik}\}$, $i =$ pattern #, $k =$ output #

• Neural network:
  – output unit value: $o_i = \{o_{ik}\}$, $i =$ pattern #, $k =$ output #
  – Also sometimes write $o_{ik}$ as $f_{\text{network},k}(x_i)$ to indicate that the neural network output is a function of the input.
Learning in Neural Networks

• For a single layer feedforward neural network, learning can be seen as shifting the decision boundary until the training set examples are classified correctly
• Decision boundary is shifted by changing the weights (including the bias)
• Learning is a way to automate the search process.

\[ W_{jk,\text{new}} = W_{jk} + \Delta W_{jk} \]
How to find useful weights

- Labelled training data. For a given set of weights, the network will make predictions in response to inputs.
- Classification: count the number of correct responses
- Regression: work out the squared error, i.e:

\[
E = \sum_{i=1}^{n} \sum_{k=1}^{m} E_{ik} = \sum_{i=1}^{n} \sum_{k=1}^{m} [y_{ik} - f_{\text{network},k}(x_i)]^2
\]

where \( n \) is the number of training patterns (observations), \( m \) is the number of outputs, \( y_{ik} \) is the \( k \)th correct output label for pattern \( i \) and \( f_{\text{network},k}(x) \) is the \( k \)th element of the overall neural network function.
Finding useful weights

• Use optimisation to find the set of weights which minimises \( E \)
• Global optimum set of weights would have for every weight:

\[
\frac{\partial E}{\partial w} = 0
\]

• Unfortunately, we can't solve that equation
• Let's now ignore the state space, we need to search in the weight space (which is typically a lot bigger)
• The partial derivatives with respect to each weight give us a direction of steepest descent in our quest to minimise \( E \)
Perceptron Learning Rule (1)

- Adjusting weights to minimise the error at the output of the network
- \( y_k \) = target of output \( k \)
- \( o_k \) = actual output \( k \)
- \( x_j \) = input \( j \)
- \( w_{jk} \) = weight between input \( j \) and output \( k \)

![Diagram](attachment:image.png)
Perceptron Learning Rule (2)

- If $o_k = y_k$
  - No weight change needed
- If $o_k = 1$, $y_k = 0$, $y_k - o_k = -1$
  - Weighted input to output $k$ is too large
    \[ \Delta w_{jk} = -\eta x_j \]
- If $o_k = 0$, $y_k = 1$, $y_k - o_k = +1$
  - Weighted input to output $k$ is too small
    \[ \Delta w_{jk} = +\eta x_j \]
- Gives us:
  \[ \Delta w_{jk} = (y_k - o_k)\eta x_j \]
Learning: guided by performance

• Measure of performance: sum squared error

\[
E = \sum_{i=1}^{n_{tr}} \sum_{k=1}^{m} E_{ik} = \sum_{i=1}^{n_{tr}} \sum_{k=1}^{m} [y_{ik} - f_{\text{network},k}(x_i)]^2
\]

• \( y_{ik} \) is the \( k \)th correct output label (value) for training pattern \( i \)

• \( f_{\text{network},k}(x) \) is the \( k \)th element of the overall neural network function (i.e. \( k \)th output \( o_{ik} \))
Learning: error rates

\[ E = \sum_{i=1}^{n_{tr}} \sum_{k=1}^{m} E_{ik} = \sum_{i=1}^{n_{tr}} \sum_{k=1}^{m} \left[ y_{ik} - f_{\text{network},k}(x_i) \right]^2 \]

• Can divide \( E \) by the number of training patterns \( n_{tr} \) to get average squared error per observation

• Textbook has

\[ E = \frac{1}{2} \sum_{i=1}^{n_{tr}} \sum_{k=1}^{m} E_{ik} \]

• Constant multipliers don’t really matter – absorbed in learning rate \( \eta \) during the weight update:

\[ w = w - \eta \frac{\partial E}{\partial w} = w + \Delta w \]
Learning: optimisation

• The neural network models the training set best when the error is minimised

• Parameters:
  – weights
  – biases
  – (number of hidden units)

• Assume that each output is equally important
  – Could weight the errors from different outputs
Optimisation Algorithms

- Gradient descent
- $2^{nd}$ order methods
- Evolutionary algorithms
- Expectation-maximisation algorithm
- Simulated annealing
Gradient Descent

• Gradient is often available, so it makes sense to use it (follow the gradient downhill)
Numerical derivation

\[ \Delta w \propto -\frac{\partial E}{\partial W} \]

Error surface

\[ \frac{dE}{dw} = \frac{E(0.454) - E(0.452)}{0.002} \]

Minima
Basic gradient descent learning

• Minimise the error using the following:
  – Choose a random set of initial weights, e.g. from $U[-0.5,0.5]$ (uniform distribution)
  – Repeat:
    • Run an input training pattern through the network and record the activations of hidden units and output units
    • Calculate $E$ for the current weights.
    • Update every weight via
      
    – Until stop condition, e.g. happy with performance or nothing much changing

• $\eta > 0$ is the learning rate, typically 0.1
Gradient Descent in Single Layer Neural Networks

$$w_{jk, new} = w_{jk} + \eta \times E_k \times f'(z_k) \times x_j$$

- $w_{jk, new}$ is the updated weight between input $j$ and output $k$
- $w_{jk}$ is the current weight between input $j$ and output $k$
- $\eta$ is the learning rate
- $E_k$ is the error for output $k$
  $$E_k = (y_k - f(z_k))$$
- $z_k$ is the weighted input to output $k$

$$f'(z_k) = \sum_{j=0}^{n} x_j w_{jk}$$

For the sigmoid function $f'(z_k) = f(z_k)(1-f(z_k))$

- $x_j$ is the activation of the input $j$
Single layer: continuous activation function

- The neural network has just one layer of weights – no hidden units

\[
\frac{\partial E}{\partial w_{ab}} = \sum_{i=1}^{n_r} \sum_{k=1}^{m} \frac{\partial}{\partial w_{ab}} \left[ y_{ik} - f_{\text{network},k}(x_i) \right]^2
\]

\[
= \sum_{i=1}^{n_r} \sum_{k=1}^{m} -2[ y_{ik} - f_{\text{network},k}(x_i) ] \frac{\partial f_{\text{network},k}(x_i)}{\partial w_{ab}}
\]

Assume sigmoid function: 
\[
f_{\text{network},k}(x_i) = \frac{1}{1 + e^{-\sum_{j=0}^{p} w_{jk} x_{ij}}}
\]

- Note: including bias weights as \( w_{0k} \) (j=0): \( x_{i0} = 1 \) for all training patterns \( i \)
Single layer: continuous activation function

- So:

\[
\frac{\partial f_{\text{network},k}(x_i)}{\partial w_{ab}} = \begin{cases} 
-1 - 1x_{ia} & , b = k \\
- \sum_{j=0}^{p} w_{jk}x_{ij} & 1 + e \\
0, b \neq k 
\end{cases}
\]

So

\[
\frac{\partial E}{\partial w_{ab}} = -2 \sum_{i=1}^{n_t} \left[ y_{ib} - f_{\text{network},b}(x_i) \right] x_{ia} \left( - \sum_{j=0}^{p} w_{jk}x_{ij} \right)^2 \\
= -2 \sum_{i=1}^{n_t} \left[ y_{ib} - o_{ib} \right] o_{ib}^2 x_{ia}
\]

Thus \(\Delta w_{ab} = 2\eta \sum_{i=1}^{n_t} \left[ y_{ib} - o_{ib} \right] o_{ib}^2 x_{ia}\)
Summary
Russell and Norvig, Chapter 18 (18.7)

• Applications
• History
• Units
• Decision Boundaries
• Learning
Learning in neural networks

• Single-layer networks
• Multi-layer networks
• Outputs:
  – Binary output: function = threshold
  – Continuous output: threshold = sigmoid
Bias weights

• Rather than use separate notation for bias weights, we use $w_{0b}$ for the bias weight on a hidden layer unit and $w_{0c}$ for the bias weight on an output unit.

• To make this work, we assume that there is an extra input $x_{i0}$ and an extra hidden unit $h_{i0}$, both of which are permanently fixed at +1.
Learnin in Neural Networks

• For a single layer feedforward neural network, learning can be seen as shifting the decision boundary until the training set examples are classified correctly

• Decision boundary is shifted by changing the weights (including the bias)

\[ W_{jk, new} = W_{jk} + \Delta W_{jk} \]
Perceptron Learning Rule (1)

- Adjusting weights to minimise the error at the output of the network
- $y_k =$ target of output $k$
- $o_k =$ actual output $k$
- $x_j =$ input $j$
- $w_{jk} =$ weight between input $j$ and output $k$

Target = $y_k$

Input $j$          Output $k$
Learning in Multi-layer Perceptrons

• Multi-layer networks (with a hidden layer) can approximate any function (including those that are not linearly separable)

• Accuracy depends upon
  – how complex the true function mapping input to output is
  – amount of data available (tells us about this function)
  – number of hidden units
  – effectiveness of learning, ie: optimisation of the weights
Backpropagation

• Backpropagation works by gradient descent of the error function in the space of network weights
• For continuous units, error function is differentiable
Weight Space (1)

- There are a lot of weights in a neural network.
- With a single hidden layer, every input connected to every hidden unit and every hidden unit connected to every output unit + biases on all units, there are \((p+1)l + (l+1)m\) weights.
- For example: NETtalk had \(p=7\times29=203\) inputs, \(l=80\) hidden units, \(m=26\) output units.
- So that is: \(204\times80+81\times26 = 18,426\) weights.
- We are trying to search for a global minimum of \(E\) over that space – barely imaginable.

```
Weight Space (1)
```

```
\begin{align*}
  \mathbf{w} &= (x_1, h_1, o_1, h_2, o_2, \ldots, x_p, h_l, o_m)
\end{align*}
```
From one perspective, you can never get enough data to accurately determine so many weight values. But many near-optimal solutions exist which will do quite well in practice. These can be found based on limited data, and with simple optimisation methods like gradient descent. It is useful to use a smaller network where possible (fewer hidden units):

- Easier to train
- Likely better generalisation
Backpropagation strategy

• Networks with hidden nodes can be trained – non-linearly separable problems can be learned

• The overall strategy is:
  – Calculations based on errors at the output are propagated backwards through the network
  – Each hidden node receives some “blame” based on its contribution to the output error
  – As a result, hidden activations come to represent higher order features of the input, useful for the learning task
1. Feed forward
2. Error calculation
3. Weight modification

Feed forward

\[ g(u) = \frac{1}{1 + e^{-u}} \]
\[ u_{iq} = \sum_{j=0}^{p} w_{jq} x_{ij} \]
\[ h_{iq} = g(u_{iq}) \]
\[ z_{ik} = \sum_{q=0}^{l} w_{qk} h_{iq} \]
\[ o_{ik} = g(z_{ik}) \]

Error calculation

\[ E = \sum_{i=1}^{n_{ir}} \sum_{k=1}^{m} E_{ik} \]
\[ E_{ik} = \left[ y_{ik} - o_{ik} \right]^2 \]

Weight modification

\[ w = w_{old} + \Delta w \]
\[ \Delta w = -\eta \frac{\partial E}{\partial w} \]
Backpropagation algorithm for learning in multilayer networks

**function** BACK-PROP-LEARNING(*examples, network*) **returns** a neural network

**inputs:** *examples*, a set of examples, each with input vector \( \mathbf{x} \) and output vector \( \mathbf{y} \)

*network*, a multilayer network with \( L \) layers, weights \( W_{j,i} \), activation function \( g \)

repeat
   repeat
      for each \( e \) in *examples* do
         for each node \( j \) in the input layer do
            \( a_j \leftarrow x_j[e] \)  \hspace{2cm} **Set the input activations**
         for \( l = 2 \) to \( L \) do
            \( \text{in}_i \leftarrow \sum_j W_{j,i} a_j \)  \hspace{2cm} **For each layer, feedforward the activations**
            \( a_i \leftarrow g(\text{in}_i) \)
         for each node \( i \) in the output layer do
            \( \Delta_i \leftarrow g'(\text{in}_i) \times (y_i[e] - a_i) \)  \hspace{2cm} **For each output, determine the error**
         for \( l = L-1 \) down to \( 1 \) do
            for each node \( j \) in layer \( l \) do
               \( \Delta_j \leftarrow g'(\text{in}_j) \sum_i W_{j,i} \Delta_i \)  \hspace{2cm} **For each layer, determine the ‘blame’ attributed to each unit and work out the new weights**
      until some stopping criterion is satisfied
   return NEURAL-NET-HYPOTHESIS(*network*)

Repeat until the performance has reached the desired level, or until the max training time.
Backpropagation Derivation (1)

Assume logistic unit activation function \( g() \)
Weight update rule for hidden-output weights \( w_{bc} \) (includes output unit bias weights), \( b \in 0,..,l \), \( c \in 1,..,m \):

\[
\Delta w_{bc} = -\eta \frac{\partial E}{\partial w_{bc}} = -\eta \frac{\partial}{\partial w_{bc}} \sum_{i=1}^{n_r} \sum_{k=1}^{m} E_{ik} = -\eta \sum_{i=1}^{n_r} \frac{\partial E_{ic}}{\partial w_{bc}} \\
\frac{\partial E_{ic}}{\partial w_{bc}} = \frac{\partial [y_{ic} - o_{ic}]^2}{\partial w_{bc}} = -2[y_{ic} - o_{ic}] \frac{\partial o_{ic}}{\partial w_{bc}}.
\]

\( o_{ic} = g(z_{ic}) = g(\sum_{q=0}^{l} w_{qc} h_{iq}). \)

\( g(z_{ic}) = \frac{1}{1 + e^{-z_{ic}}}, \) so \( \frac{\partial g(z_{ic})}{\partial w_{bc}} = \frac{e^{-z_{ic}}}{(1 + e^{-z_{ic}})^2} \frac{\partial z_{ic}}{\partial w_{bc}} = e^{-z_{ic}} o_{ic}^2 h_{ib} \)

So \( \Delta w_{bc} = 2\eta \sum_{i=1}^{n_r} [y_{ic} - o_{ic}]o_{ic}^2 e^{-z_{ic}} h_{ib} \)

If \( b = 0 \) (bias weight on an output unit), \( h_{i0} = 1 \) in above equation.
Backpropagation Derivation (2)

Weight update rule for input-hidden layer weight $w_{ab}$ (includes hidden unit bias weights), $a \in 0, \ldots, p$, $b \in 1, \ldots, l$:

$$\Delta w_{ab} = -\eta \frac{\partial E}{\partial w_{ab}} = -\eta \frac{\partial}{\partial w_{ab}} \sum_{i=1}^{n_r} \sum_{k=1}^{m} E_{ik} = -\eta \sum_{i=1}^{n_r} \sum_{k=1}^{m} \frac{\partial E_{ik}}{\partial w_{ab}}$$

$$\frac{\partial E_{ik}}{\partial w_{ab}} = \frac{\partial [y_{ik} - o_{ik}]^2}{\partial w_{ab}} = -2[y_{ik} - o_{ik}] \frac{\partial o_{ik}}{\partial w_{bc}}.$$

$$o_{ik} = g(z_{ik}) = g\left(\sum_{q=0}^{l} w_{qk} h_{iq}\right).$$

$$g(z_{ik}) = \frac{1}{1 + e^{-z_{ik}}}, \text{ so } \frac{\partial g(z_{ik})}{\partial w_{ab}} = \frac{e^{-z_{ik}}}{(1 + e^{-z_{ik}})^2} \frac{\partial z_{ik}}{\partial w_{ab}} = e^{-z_{ik}} o_{ik}^2 w_{bk} \frac{\partial h_{ib}}{\partial w_{ab}}$$

$$\frac{\partial h_{ib}}{\partial w_{ab}} = \frac{\partial g(u_{ib})}{\partial w_{ab}} = e^{-u_{ib}} h_{ib}^2 \frac{\partial u_{ib}}{\partial w_{ab}}$$

$$\frac{\partial u_{ib}}{\partial w_{ab}} = \frac{\partial}{\partial w_{ab}} \sum_{j=0}^{p} w_{jb} x_{ij} = x_{ia}$$

So $\Delta w_{ab} = 2\eta \sum_{i=1}^{n_r} \sum_{k=1}^{m} [y_{ik} - o_{ik}] e^{-z_{ik}} o_{ik}^2 w_{bk} e^{-u_{ib}} h_{ib}^2 x_{ia}$
Backpropagation Implementation

• Noting the following update rules for the hidden-output and input-hidden weights, it becomes apparent what needs to be stored on the feedforward phase (producing output) so that backpropagation learning can be performed

\[
\Delta w_{bc} = 2\eta \sum_{i=1}^{n_r} [y_{ic} - o_{ic}] e^{-z_{ik}} o_{ik}^2 h_{ib} \\
\Delta w_{ab} = 2\eta \sum_{i=1}^{n_r} \sum_{k=1}^{m} [y_{ik} - o_{ik}] e^{-z_{ik}} o_{ik}^2 w_{bk} e^{-u_{ib}} h_{ib}^2 x_{ia} \\
= 2\eta \sum_{i=1}^{n_r} e^{-u_{ib}} h_{ib}^2 x_{ia} \left( \sum_{k=1}^{m} [y_{ik} - o_{ik}] e^{-z_{ik}} o_{ik}^2 w_{bk} \right)
\]

• So, in response to each training input, we need to store all the network hidden and output values and their activations (weighted input)
Learning and Generalisation

• Want the network to learn the training set and classify these examples correctly
• Also want the network to be able to generalise to a test set
• There can be a trade-off between learning and generalisation, especially when there are errors or noise in the training set
Generalization or over fitting?

Increasing the number of samples used for training reduces the problem.
Generalisation:

• Minimising error on the training set is a strategy to obtain good performance on new data
• It doesn't always work
• Possible reasons:
  – Over-fitting
  – Under-fitting
Generalisation: Over-fitting

• Model is a very good fit to the training data
• May have modelled some useful structure in the data
• Has also modelled many irrelevant aspects
• Common with overly-complex models
• Solution: early stopping
What often happens:

- This NN may be too complex for the data/problem and so prone to overfitting.
- If we stop training early, performance could be ok.
Controlling generalization by using validation data

• Dataset is divided into a training set, test set and a validation set
• Training set is used for optimizing weights
• Validation set is monitored during (or after) training for optimizing model parameters (when to stop, #nodes/weights, learning rate etc)
• Test set is used to assess how well the model performs
Generalisation: Under-fitting

• Model is not a good fit to the training data (size of E is a good indication)
• Model may be too simple
• You may not have enough data
• The data may be so noisy that you can never predict the output very accurately
• Solution: structure of network, quality and quantity of data
Parameter Values and Other Issues

• Weight initialization
  – Typically small random values with mean 0

• Learning methods
  – Batch or online updates

• Learning rate
  – $0 < \eta \leq 1$, typically 0.1

• Stopping criterion

• Number of layers

• Number of nodes in hidden layers

• Number of examples
Learning methods

• Batch learning
  • Initialise the weights
  • Repeat as necessary:
    • Process all training data
    • Update weights

• Online learning
  • Initialise the weights
  • Repeat as necessary:
    • Process one training example
    • Update weights
Batch learning

• Requires us to feed in all $n_{tr}$ training patterns and record hidden and output unit activations and outputs

• Batch NN learning tends to follow: large-scale, medium-scale, then small-scale optimisation
  • If misled early on, might never recover
  • Rare patterns might be missed
Online learning (1)

• Sometimes we have a lot of data, e.g. >5000 bitmap patterns for Assignment 2

• If doing e.g. 10,000 epochs (iterations) of learning, this results in a lot of calculations

• Alternative: online mode:
  • At each iteration, choose 1 training pattern at random, and update based on this alone

• Is implemented in Assignment 2 supplied code
Online learning (2)

• Advantages:
  • feedforward phase is $n_{tr}$ times quicker
  • Can escape local minima due to constant shifting of targets
  • Eventually sees all training patterns and does the right thing “on average”
  • Can cope with non-stationary targets, ie: if the input-output function is changing over time, this method can react to changing training data
Online learning (3)

• Disadvantages:
  • May have trouble converging to a solution: current solution is not optimal for any single training pattern
  • In practice, seems to perform quite well, but probably needs more iterations than batch learning
Building a Neural Network

• Specify the inputs and outputs of the problem
• Choose the simplest structure that will work
• Find appropriate connection weights
• Check the network works on the training data, and test generalisation on test data
• If performance is not good enough, improve one of these features
Learning to diagnose heart disease (Thornton, 1993)

- A database with 300 records which contain attributes for individuals diagnosed as either being sick or healthy

<table>
<thead>
<tr>
<th>age</th>
<th>sex</th>
<th>pain</th>
<th>b.pres</th>
<th>chol</th>
<th>b.sug</th>
<th>ECG</th>
<th>h.rate</th>
<th>angina</th>
<th>oldpeak</th>
<th>slope</th>
<th>vessels</th>
<th>thal</th>
<th>diag</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>male</td>
<td>asymt</td>
<td>112</td>
<td>290</td>
<td>fal</td>
<td>hyp</td>
<td>153</td>
<td>fal</td>
<td>0.0</td>
<td>up</td>
<td>1</td>
<td></td>
<td>norm     sick</td>
</tr>
<tr>
<td>41</td>
<td>male</td>
<td>abnang</td>
<td>120</td>
<td>157</td>
<td>fal</td>
<td>norm</td>
<td>182</td>
<td>fal</td>
<td>0.0</td>
<td>up</td>
<td>0</td>
<td></td>
<td>norm     well</td>
</tr>
<tr>
<td>67</td>
<td>fem</td>
<td>notang</td>
<td>115</td>
<td>564</td>
<td>fal</td>
<td>hyp</td>
<td>160</td>
<td>fal</td>
<td>1.6</td>
<td>flat</td>
<td>0</td>
<td></td>
<td>rev      well</td>
</tr>
</tbody>
</table>

.
Preparing data for training

• The task is to classify attribute values of an individual as sick or healthy
• Convert attribute values to input and output unit values preferably without imposing any phoney similarity relations between them
• Nominal attributes: separate binary nodes?
• Numeric attributes: normalize?
• Missing values: Use mean of existing values? Use 0? Base on values in similar patterns (imputation)?

<table>
<thead>
<tr>
<th>age</th>
<th>sex</th>
<th>pain</th>
<th>b.pres</th>
<th>chol</th>
<th>b.sug</th>
<th>ECG</th>
<th>h.rate</th>
<th>angina</th>
<th>oldpeak</th>
<th>slope</th>
<th>vessels</th>
<th>thal</th>
<th>diag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.44</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.12</td>
<td>0.290</td>
<td>1</td>
<td>0.84</td>
<td>1</td>
<td>0.0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0.41</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.20</td>
<td>0.157</td>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>0.0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.67</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.15</td>
<td>0.564</td>
<td>1</td>
<td>0.88</td>
<td>1</td>
<td>1.6</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Encoding $v$ output classes

• Simplest option: $v$ outputs: for each: 1 means it is of that class, 0 means it isn’t
• Treats each class equally
• The network is never completely sure which output class it is seeing, and can produce a number of output values in the range (0,1)
• Choose the output with highest value as the prediction
• Could interpret output values as probabilities by using a normalising constant
The network

- Choose a network topology
- Determine the number of hidden units, if any
- Create network
  \[ \text{NN1}(\text{int}\ n\text{Input}, \text{int}\ n\text{Output}, \text{int}\ seed) \]
  \[ \text{NN1}(\text{String}\ \text{filename}) \]
- Learning
  double \text{train}(\text{double}[]\ x, \text{double}[]\ d, \text{double}\ \text{eta})
- Evaluate the network configuration by running tests
  double[] \text{feedforward}(\text{double}[]\ x)
Evaluating the network

• Record generalization performance on different test sets
  • Provides no model-specific explanation
• Inspecting internal activation
  • Provides knowledge on the effect of different inputs on the model
• Studying network weights
  • Provides knowledge on decisions made in the model
Testing different inputs

Network 1 (seed 1): SSE=0.001, 89% correct on test set
Network 2 (seed 2): SSE=0.021, 91% correct on test set
Network 3 (seed 3): SSE=0.005, 77% correct on test set
Network 1 has trouble with $p_1$ and $p_2$. What is special about them?
Inspecting internal activation

Visualize (spatial) similarity between internal activations
Studying weights

Hinton diagram visualizes information how weights are used, which units are relevant

**hintonw in Matlab**
Tutorial 8

• Current best learning
• Decision Trees
• Naive Bayes Model
Current Best Learning

- Recognising true and false positives and negatives
  - False positive – hypothesis predicts positive but actually negative
  - False negative – hypothesis predicts negative but actually positive
  - True positive – hypothesis predicts positive and actually positive
  - True negative – hypothesis predicts negative and actually negative
Current Best Learning

• How to change the hypothesis when the example is:
  • False positive -> specialise
  • False negative -> generalise
  • True positive -> no change
  • True negative -> no change
Decision Trees

• Determine the ‘most important’ attribute using information theory

• Recommended steps:
  • State the positive and negative examples for each attribute
  • Calculate remainder using equation, showing all working (if you use the graph, state that you have done so)
  • Calculate gain using equation (may not be needed)
  • Determine ‘most important’ attributed given remainders or gains that have been calculated
Decision Trees

• Completing the tree
• In examples with small data sets, information theory should be used to determine the top attribute for the tree, but the remaining branches may be simple enough to determine by looking at the data and finding an attribute that classifies the remaining examples
• Nodes should be attributes
• Leaf nodes should be outcomes
• Branches should be labelled with the attribute value
Naive Bayes

• For each test pattern, determine the relative probabilities for each of the possible outcomes
  • State the equation being used
  • State the conditional probabilities being used
  • Calculate the relative probabilities

• If all you need is the result of the model, state the outcome predicted by the model (which outcome has a higher relative probability)

• If you also need the normalised probabilities, the relative probabilities should be adjusted so that they add up to 1
Naive Bayes
Calculating Conditional Probabilities

- \( P(\text{Hungry} = \text{true} \mid \text{Will Wait} = \text{false}) = \frac{P(\text{Hungry} = \text{true} \text{ and Will Wait} = \text{false})}{P(\text{Will wait} = \text{false})} = \frac{2/10}{5/10} = 2/5 \)

<table>
<thead>
<tr>
<th>Fri/Sat</th>
<th>Hungry</th>
<th>Patrons</th>
<th>Type</th>
<th>Will wait?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FALSE</td>
<td>TRUE</td>
<td>Some</td>
<td>French</td>
</tr>
<tr>
<td>2</td>
<td>FALSE</td>
<td>TRUE</td>
<td>Full</td>
<td>Thai</td>
</tr>
<tr>
<td>3</td>
<td>FALSE</td>
<td>FALSE</td>
<td>Some</td>
<td>Burger</td>
</tr>
<tr>
<td>4</td>
<td>TRUE</td>
<td>TRUE</td>
<td>Full</td>
<td>Thai</td>
</tr>
<tr>
<td>5</td>
<td>TRUE</td>
<td>FALSE</td>
<td>Full</td>
<td>French</td>
</tr>
<tr>
<td>6</td>
<td>FALSE</td>
<td>TRUE</td>
<td>Some</td>
<td>Italian</td>
</tr>
<tr>
<td>7</td>
<td>FALSE</td>
<td>FALSE</td>
<td>None</td>
<td>Burger</td>
</tr>
<tr>
<td>8</td>
<td>FALSE</td>
<td>TRUE</td>
<td>Some</td>
<td>Thai</td>
</tr>
<tr>
<td>9</td>
<td>TRUE</td>
<td>FALSE</td>
<td>Full</td>
<td>Burger</td>
</tr>
<tr>
<td>10</td>
<td>TRUE</td>
<td>TRUE</td>
<td>Full</td>
<td>Italian</td>
</tr>
</tbody>
</table>
Naive Bayes
Calculating Conditional Probabilities

• \( P(\text{Hungry} = \text{true} \mid \text{Will Wait} = \text{false}) = \frac{2}{5} \)

<table>
<thead>
<tr>
<th>Fri/Sat</th>
<th>Hungry</th>
<th>Patrons</th>
<th>Type</th>
<th>Will wait?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>FALSE</td>
<td>TRUE</td>
<td>Full</td>
<td>Thai</td>
</tr>
<tr>
<td>5</td>
<td>TRUE</td>
<td>FALSE</td>
<td>Full</td>
<td>French</td>
</tr>
<tr>
<td>7</td>
<td>FALSE</td>
<td>FALSE</td>
<td>None</td>
<td>Burger</td>
</tr>
<tr>
<td>9</td>
<td>TRUE</td>
<td>FALSE</td>
<td>Full</td>
<td>Burger</td>
</tr>
<tr>
<td>10</td>
<td>TRUE</td>
<td>TRUE</td>
<td>Full</td>
<td>Italian</td>
</tr>
</tbody>
</table>
Summary

• Units
• Decision Boundaries
• Logic Functions
• Single and Multi Layer Networks
• Learning
  • Gradient Descent
  • Backpropagation
• Generalisation
• Learning methods
  • Batch
  • Online
• Building a Neural Network
Assignment 3

Handwritten letter recognition
Due 5pm Friday 25 October 2013
(no penalty extension to 5pm Monday 4/11)
data.txt
Section A: Basic and Advanced Classifier (10 marks)
Divide the data set into at least two sets (training and test), adding extra data as necessary. Investigate and optimise the performance of the already implemented single layer neural network. Investigate and optimise the performance of the already implemented ID3 decision tree. Implement an advanced classifier using your chosen technique, and investigate and optimise its performance. Discuss the differences in performance obtained for the optimised single-layer neural network, the optimised ID3 decision tree, and your optimised advanced classifier.

Section B: Refinements (capped at 10 marks)
Any number (0 to 4) of the parts (1-4) may be completed. All are assigned the same marks. Describe your implementation of the method, including references to published literature, provide correct implementation of the method, and describe the performance on training and test data, especially in comparison to the performance without the task implemented.

Bonus Marks (up to 2 bonus marks available)
Complete your assignment individually
Submit by 25th October