Recap: Words and Rules

- Does the theory of words and rules explain the dichotomy between regular and irregular verbs?
- Is SPE a plausible theory of how the past tense is formed?
- What does evidence from language development tell us about regular and irregular verbs?
- Maybe a rule is not necessary to explain the past tense.
- Maybe children simply analogise from verbs they already know (e.g., from correct forms like folded, molded, scolded to over-regularisations like holded).
- All-rules versus all-memory approach.

A Single Neuron

- Neuron receives inputs and combines these in the cell body.
- If the input reaches a threshold, then the neuron may fire (produce an output).
- Some inputs are excitatory, while others are inhibitory.
Biological Neural Networks

- In biological neural networks, connections are synapses.
- **Input connection** is conduit through which a member of a network **receives** information (INPUT).
- **Output connection** is a conduit through which a member of a network **sends** information (OUTPUT).

Connectionism

Connectionism is the name for a **computer modeling** approach based on how information processing occurs in neural networks (connectionist networks are called **artificial neural networks**).

Anatomy of a Connectionist Model

Units are to a connectionist model what neurons are to a biological neural network — the basic information processing structures.

Biological neural networks are organized in **layers of neurons**. Connectionist models are organized in layers of units, not random clusters.

But what you see here still isn’t a network. Something is missing. **Network connections** are conduits through which information flows between members of a network.
Anatomy of a Connectionist Model

Connections are represented with lines. Arrows in a connectionist model indicate the flow of information from one unit to the next.

Perceptron: An Artificial Neuron

Perceptron was developed by Frank Rosenblatt in 1957 and can be considered as the simplest artificial neural network.

\[ u(x) = \sum_{i=1}^{n} w_i x_i \]

\[ y = f(u(x)) = \begin{cases} 1, & \text{if } u(x) > \theta \\ 0, & \text{otherwise} \end{cases} \]

Activation state: 0 or 1 (-1 or 1)

Inputs are in the range [0, 1], where 0 is “off” and 1 is “on”.

Weights can be any real number (positive or negative).
**Perceptrons for Logic**

**Perceptron for AND**

- **Function:** \( f(x_1, x_2) = \begin{cases} 1 & \text{if } \sum x_i \geq \theta \text{ then } 1 \text{ else } 0 \\ 0 & \text{otherwise} \end{cases} \)
- **Example:**
  \[
  0 \cdot 0.5 + 1 \cdot 0.5 = 0.5
  \]

**Perceptron for OR**

- **Function:** \( f(x_1, x_2) = \begin{cases} 1 & \text{if } \sum x_i \geq \theta \text{ then } 1 \text{ else } 0 \\ 0 & \text{otherwise} \end{cases} \)
- **Example:**
  \[
  0 \cdot 0.5 + 1 \cdot 0.5 = 0.5
  \]

**Perceptron for XOR**

- **Function:** \( f(x_1, x_2) = \begin{cases} 1 & \text{if } \sum x_i \geq \theta \text{ then } 1 \text{ else } 0 \\ 0 & \text{otherwise} \end{cases} \)
- **Example:**
  \[
  0 \cdot 0.5 + 1 \cdot 0.5 = 0.5
  \]
  
  **XOR** is an exclusive OR because it only returns a true value of 1 if the two values are exclusive, i.e., they are both different.
### Perceptrons for Logic

#### Perceptron for XOR

- **Symbolic Representation:**
  
  \[
  f(x_1, x_2) = \begin{cases} 
  1 & \text{if } \sum x_i \geq \theta \\
  0 & \text{else} 
  \end{cases}
  \]

  where \( x_1, x_2 \) are the inputs, \( \theta \) is the threshold.

- **Example Calculation:**
  
  \[
  0 \cdot 0.5 + 0 \cdot 0.5 = 0 
  \]

#### Perceptrons as Classifiers

Perceptrons are **linear classifiers**, i.e., they can only separate points with a **hyperplane** (a straight line).

### What is the Perceptron Really Seeing?

Sequence of exemplars presented to the Perceptron:

<table>
<thead>
<tr>
<th>N</th>
<th>input ( x )</th>
<th>target ( t )</th>
<th>output ( o )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0,1,0,0)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>(1,0,0,0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>(0,1,1,1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>(1,0,1,0)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>(1,1,1,1)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>(0,1,0,0)</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- The above Perceptron has 4 inputs (binary) \( \approx \) feature vector representing each exemplar.
- The Perceptron see 6 exemplars or training items.
- We know what the right answer is \( \approx \) output.
- What would happen if we used random weights/threshold?
Learning

Q1: But... choosing weights and threshold $\theta$ for the perceptron is not easy! How to learn the weights and threshold from examples?

A1: We can use a learning algorithm that adjusts the weights and threshold based on examples.

http://www.youtube.com/watch?v=vGwemZhPlsA&feature=youtu.be

Learning Rule

Learning happens by adjusting weights. The threshold can be considered as a weight.

Perceptron’s Learning Rule

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

- $\eta, 0 < \eta \leq 1$ is a constant called learning rate.
- $t$ is the target output of the current example.
- $o$ is the output obtained by the Perceptron.
**Learning Rule**

**Perceptron's Learning Rule**

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

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

- \( o = 1 \) and \( t = 1 \) \( \Delta w_i = \eta(t - o)x_i = \eta(1 - 1)x_i = 0 \)
- \( o = 0 \) and \( t = 1 \) \( \Delta w_i = \eta(t - o)x_i = \eta(1 - 0)x_i = \eta x_i \)

- Learning rate \( \eta \) is positive; controls how big changes \( \Delta w_i \) are.
- If \( x_i > 0 \), \( \Delta w_i > 0 \). Then \( w_i \) increases in an attempt to make \( w_i x_i \) become larger than \( \theta \).
- If \( x_i < 0 \), \( \Delta w_i < 0 \). Then \( w_i \) reduces.

**Learning Algorithm**

1. Initialize all weights randomly.
2. repeat
3. for each training example do
   4. Apply the learning rule.
5. end for
6. until the error is acceptable or a certain number of iterations is reached

This algorithm is guaranteed to find a solution with error zero in a limited number of iterations as long as the examples are linearly separable.

**Summary**

- What does this have to do with the words versus rules debate?
  - Connectionism is a computer modeling approach inspired by neural networks.
  - Anatomy of a connectionist model: units, connections
  - The Perceptron as a linear classifier.
  - A learning algorithm for Perceptrons

**Next lecture:** multilayer perceptrons (neural networks).

---

**Learning Rule: Exercise**

**Perceptron's Learning Rule**

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

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

Consider a Perceptron with only one input \( x_1 \), weight \( w_1 = 0.5 \), threshold \( \theta = 0 \) and learning rate \( \eta = 0.6 \). Consider also the training example \( \{ x_1 = -1, t = 1 \} \). For now, let's temporarily ignore the learning of the threshold and consider it fixed.

- Determine the output of the Perceptron for the input -1:
  \( w_1x_1 = 0.5(-1) = -0.5 \leq \theta \rightarrow o = 0 \)
- The new weight \( w_1 \) after applying the learning rule:
  \( \Delta w_1 = 0.6(1 - 0)(-1) = -0.6 \rightarrow w_1 = 0.6 - 0.6 = -0.1 \)
- The new output of the Perceptron for the input -1:
  \( w_1x_1 = -0.1(-1) = 0.1 \geq \theta \rightarrow o = 1 \)