**Artificial neural networks, simple supervised learning**

**AIMS**

- Understand the aim of using neural networks for pattern classification.
- Describe how a set of examples of stimuli and correct responses can be used to train an artificial neural network to respond correctly via changes in synaptic weights governed by the firing rates of the pre- and post-synaptic neurons and the correct post-synaptic firing rate.
- Describe how this type of learning rule is used to perform pattern recognition in a perceptron.
- Discuss the limitation of the perceptron.

**READING**

Books 1, 2, 5.

---

**recap: Learning in neural networks**

**The problem:** find connection weights such that the network does something useful.

**Solution:**

Experience-dependent learning rules to modify connection weights, i.e. learn from examples.

1. ‘Unsupervised’ (no ‘teacher’ or feedback about right and wrong outputs)
2. ‘Supervised’:
   - A. Evolution/genetic algorithms
   - B. Occasional reward or punishment (‘reinforcement learning’)
   - C. Fully-supervised: each example includes correct output.
Consider the network with weights \( w_1 \) and \( w_2 = 1 \), threshold \( T=1.5 \) and a threshold logic transfer function.

\[
\text{o} = f(h), \ h = w_1 x_1 + w_2 x_2
\]

The line separating \( o=1 \) from \( o=0 \) is where the net input equals the threshold: \( w_1 x_1 + w_2 x_2 = T \), giving line: \( x_2 = -(w_1/w_2)x_1 + T/w_2 \) (i.e. \( y = mx+c \) format).

Changing the weights changes the line.

So learning to classify two groups of input patterns means finding weights so that the line separates the groups.

Pattern classification: linear separation of categories

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( h )</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2.0</td>
<td>1</td>
</tr>
</tbody>
</table>

It performs the logical 'AND' function

\[
\text{o} = \begin{cases} 1 & \text{for men} \\ 0 & \text{for women} \end{cases}
\]

Pattern classification, example

'Learning' or 'training' on example patterns is used to define the weights and thus the 'decision region'. Performance depends on how well it 'generalises' i.e. how well it classifies new data. These networks perform 'linear discrimination': the decision boundary is always linear.
Reminder: the ‘Hebb rule
\[ w_{ij} \rightarrow w_{ij} + \varepsilon x_j o_i \]

Supervised learning: the ‘delta rule’

- Given ‘training set’ of inputs \( x^n \) \((x_1^n, x_2^n, \ldots)\) with target output values \( t^n \) \((t_1^n, t_2^n, \ldots)\) where \( n=1,2,\ldots \)
- Present input pattern \( n \) and find the corresponding output values \( o^n \) \((o_1^n, o_2^n, \ldots)\)
- Change connection weights according to:
  \[ w_{ij} \rightarrow w_{ij} + \varepsilon x_j^n (t_i^n - o_i^n) \]
- Present the next input pattern: \( n \rightarrow n+1 \)

The term \((t_i^n - o_i^n)\) is also known as ‘delta’ \( \delta^n \): the ‘error’ made by output \( i \) for input pattern \( n \). The delta rule changes weights so as to reduce this error.

The perceptron (Rosenblatt, 1962)

output =1 if pattern detected (threshold logic function)

Present input patterns \( x^n \) (some are pattern to be detected with target \( t^n = 1 \), others are ‘foils’ to be ignored with target \( t^n = 0 \)).

For each pattern apply the ‘delta rule’:
\[ w_j \rightarrow w_j + \varepsilon x_j^n (t^n - o^n) \]

After many presentations of the whole training set (in random order) it can find the best linear discriminator of targets from foils.

Notice that \( w_j \) changes only if \( t^n \neq o^n \) and \( x_j^n \neq 0 \).
If \( t^n > o^n \) then \( w_j \) increases; if \( t^n < o^n \) then \( w_j \) decreases, i.e. the delta rule changes weights so as to reduce the error.
Perceptrons, thresholds and multiple outputs.

Q: The delta rule only learns weights, how do we find the value for the threshold $T$?
A: Just use $T=0$ and add another input $x_0=-1$, then the weight from it $w_0$ can serve the same purpose: the condition for output to be active $w_1x_1+w_2x_2+...>T$ is the same as $w_0x_0+w_1x_1+w_2x_2+...>0$ if $x_0=-1$ and $w_0=T$ and the delta rule will find connection weight $w_0=T$.

Perceptrons can have many output units (forming a single-layer neural network) – each output is trained using the delta rule independently of the others.

Implications of using the ‘weighted sum’ of input activations II: pattern classification and linear separators.

The line separating $o=1$ from $o=0$ is where the net input equals the threshold: $w_1 x_1 + w_2 x_2 = T$, giving line: $x_2 = -(w_1/w_2)x_1 + T/w_2$ (in $y = mx+c$ format).

This line is orthogonal to the weight vector $w$, and crosses it $T/|w|$ along its length. (An input vector $x$ will activate the output if $w_1 x_1 + w_2 x_2 > T$. Since $w_1 x_1 + w_2 x_2 = |w| |x| \cos(\theta)$, this is the same as $|x| \cos(\theta) > T/|w|$)
Non-linearly separable problems

Perceptrons perform linear discrimination and can’t solve ‘non-linearly separable’ problems (Minsky & Papert, 1969)

‘Hidden’ processing units can give more power, e.g:

Multi-layer perceptrons

Q: How train connection weights if the ‘internal representation’ is not known (i.e. \(t^n\) not known for ‘hidden layer’, \(x^n_j\) not known for output?)

A: if the neurons use a smooth (continuous) transfer function, changing a connection weight from any (active) neuron anywhere in the network will change the output firing rate slightly. If the output \(o^n\) is now closer to its target value \(t^n\) then the change was good.

But note: The effect of changing a connection weight depends on the values of the errors \(\delta^n = (t^n - o^n)\), connections \(w_{ij}\) and activations \(v_i\) further up in the network.

If \(w\) decreases the error for input pattern \(n\) reduces