Artificial Neural Networks

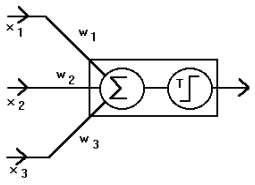
The Early Years

[From McCulloch & Pitts to Minsky & Papert]

Nick Taylor
N.K.Taylor@hw.ac.uk
EM 1.62

Model of a Neuron (1943)

- Due to McCulloch and Pitts
- Organisation



x_i are inputs, w_i are weights

- Firing Rule

$$\sum_i x_i w_i > T$$

T is the threshold

Hebb Rule (1949)

- Due to Donald Hebb
- Pre-empted by William James (1890)

- Hebb Rule

  If a particular input is always active when a neuron fires then its weight should be increased

- Interpretation

  Classical conditioning -

  When two things happen within a short time of each other, a re-occurrence of one should make the other more likely to occur

Issues with Hebb

- Problems

  a). No mechanism for weights to be decreased - they only go up!

  b). Implies a degree of responsibility at the synapse. There was little evidence for this in 1949 but it has become clear that this is the case

- Other combinations of events

		N e u r o n	
		Active	*Inactive*
I n p u t	*Active*	Hebbian Increase	Anti-Hebbian Decrease
	Inactive	?? **(**Decrease?)	?? (No Change?)

Delta Rule (1960)

- Due to Widrow and Hoff
- Also known as -
  - Widrow-Hoff Rule
  - Least Mean Square Rule
- General algorithm
  1. Determine an *error* between the desired and actual outputs
  2. Modify all of the "live" weights and the threshold so as to remove a *portion* of the error
  3. Repeat until the error is reduced to an acceptable level
- Example AND gate

Formalised Delta Rule

- We can formalise the weight adjustment rule thus -

$$\Delta^p w_{ij} = \eta \left(T_i^p - O_i^p \right) O_j^p$$

- It is often presented in another form from which it gets its name -

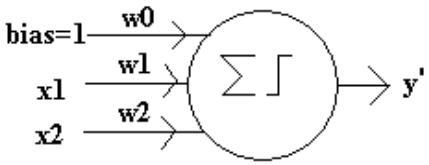
$$\Delta^p w_{ij} = \eta \delta_i^p O_j^p$$

- We don't actually need to update the weights on every presentation -

$$\Delta w_{ij} = \eta \sum_p \delta_i^p O_j^p$$

Perceptron (1957-1962)

- Due to Rosenblatt, Wightman, et al.
- Organisation



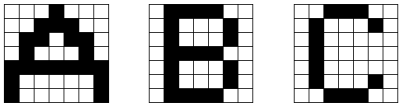
x_i are inputs, w_i are weights,
y is desired output, y' is actual output

- Firing Rule

$$\sum_i x_i w_i \geq 0$$

A Perceptron Application

- An early and influential success story of the Perceptron was recognising letters of the alphabet



- A 20x20 input grid was used
- The 400 binary input values were weighted and summed into output units representing different letters of the alphabet
  - Output 1 meant "A"
  - Output 2 meant "B"
  - Etc.

Perceptron Learning Rules

- Single output Perceptron

$$\underline{w}^{new} = \underline{w}^{old} + \eta \cdot (y - y') \cdot \underline{x}$$

- Multiple output Perceptron

$$\underline{w}^{new} = \underline{w}^{old} + \eta \cdot \Theta\left(Ns - \underline{w}^{old} \cdot \underline{z}\right) \cdot \underline{z}$$

Or

$$\Delta w_{ij} = \eta \cdot \Theta\left(Ns - y_i y_i^{'}\right) \cdot y_i x_j$$

Perceptron Convergence Theorem

- Theorem

  If a solution to a classification problem exists then the general Perceptron learning rule will find it in a finite number of steps.

- Note that this theorem is qualified by the phrase

  if a solution to a classification problem exists

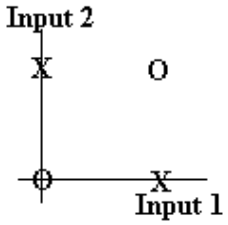
- Not all classification problems are solvable with a Perceptron
- BUT, for a problem that is solvable, the learning rule will definitely find it in a finite time

XOR Problem (I)

- Truth table

Input 1	*Input 2*	*Output*
0	0	0
1	0	1
0	1	1
1	1	0

- Linearly inseparable (Graphically)



XOR Problem (II)

- Truth table

Input 1	*Input 2*	*Output*
0	0	0
1	0	1
0	1	1
1	1	0

- Linearly inseparable (Algebraically)

0 0 0 =>	w0 < 0	(A)
1 0 1 =>	w0 + w1 >= 0	(B)
0 1 1 =>	w0 + w2 >= 0	(C)
1 1 0 =>	w0 + w1 + w2 < 0	(D)

Statement D is incompatible with Statements B and C since it requires w1+w2 to be less than a positive amount (w0 is -ve, see A) which neither is less than individually