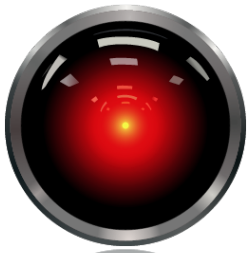


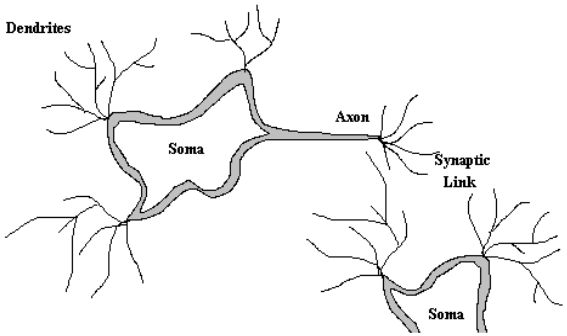
INTELLIGENT COMPUTING

(COMPUTATIONAL INTELLIGENCE)
Neural Networks Introduction

S. Sheridan

BIOLOGICAL FOUNDATIONS

- The human brain contains around 86 billion interconnected neurons. As shown in the diagram below, a typical neuron has four main components: the soma, the axon, synapse, and dendrites.

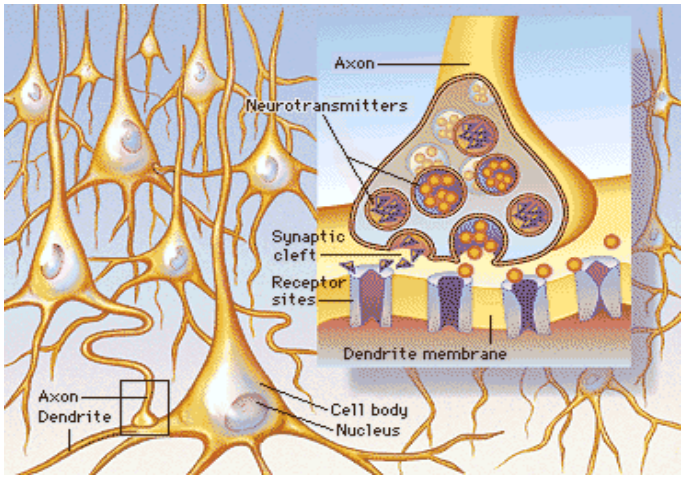


BIOLOGICAL FOUNDATIONS

- Receiving stimulus or inputs from other neurons through dendrites, the soma or the body of the cell decides when and how to respond. When the cell body receives enough stimulus or input to excite it, it transmits a response to other neurons through the axon.

- The more a link between two neurons is excited the stronger the link becomes. Hebb (1994) realised that this strengthening of the synaptic link could account for human memory. Hebb proposed that short term memory was established by a reverbitory circuit in an assembly of neurons and long term memory was based on some kind of structural change between neurons based on prolonged activity.

BIOLOGICAL FOUNDATIONS



BIOLOGICAL FOUNDATIONS

- Receiving stimulus or inputs from other neurons through dendrites, the soma or the body of the cell decides when and how to respond. When the cell body receives enough stimulus or input to excite it, it transmits a response to other neurons through the axon.

- The more a link between two neurons is excited the stronger the link becomes. Hebb (1949) realised that this strengthening of the synaptic link could account for human memory. Hebb proposed that short term memory was established by a reverbitory circuit in an assembly of neurons and long term memory was based on some kind of structural change between neurons based on prolonged activity.

BIOLOGICAL FOUNDATIONS

- From this neurophysiological research Hebb proposed his Learning Rule :

- *'When an axon of a cell (A) is near enough to excite cell (B) and repeatedly takes part in firing it, some growth process or metabolic change takes place in one or both of the cells. Such that (A)'s efficiency of firing (B) is increased.*

- A biological neural network is a complex set of these simple interconnected neurons, and their interaction with each other produces the characteristics associated with intelligence.

BIOLOGICAL FOUNDATIONS

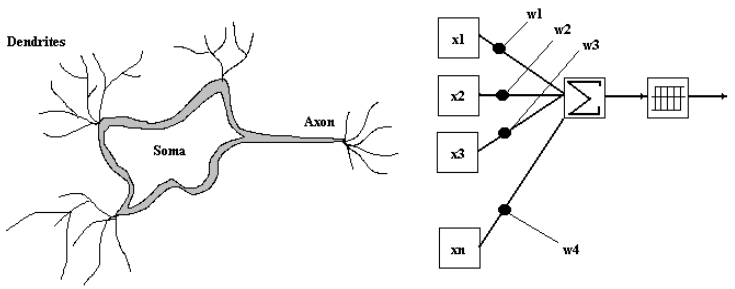
- The mechanics of biological neural networks are not fully understood, however the basic architecture of these biological constructs and their functionality can be modelled to some degree. Through the use of mathematics, electronic circuitry, or computer algorithms the interaction between neurons can be simulated and has achieved outstanding success in the field of AI.

- Many characteristics associated with human intelligence such as ***learning***, ***generalisation***, ***interpretation***, and ***prediction*** can now be duplicated in computer applications thanks to the research that has been carried out in the field of artificial neural networks.

- Where traditional computer science methods have failed neural networks have provided new ways of thinking about the problem at hand.

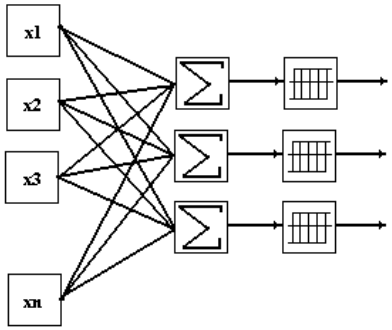
THE PERCEPTRON

- In 1962 Frank Rosenblatt invented the Perceptron which was one of the earliest neural network models. The perceptron shown in the diagram below was based on McCulloch,Pitts (1943) Binary Decision Unit (BDN), it takes a weighted sum of its inputs and sends an output of 1 if the sum is greater then its threshold value, otherwise it outputs 0. The perceptron itself consists of the weights, the summation processor, and an adjustable threshold processor.



THE PERCEPTRON

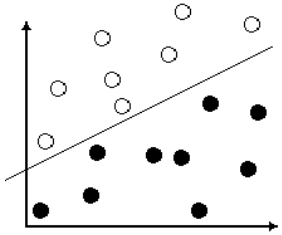
- The perceptron alone is only capable of solving toy like problems, but many perceptrons can be grouped together to solve more complex problems as shown in the diagram below. Learning in the perceptron is a process of modifying the values of the weights for each input. One amazing property of the perceptron is that whatever a perceptron can compute it can also learn to compute.



THE PERCEPTRON: LINEAR SEPARABILITY

- One drawback of the perceptron is that it can only solve problems which are linearly separable.

- The classification problem shown below is linearly separable because a line can be drawn to separate both output categories. The line separating the two categories is sometimes called the decision surface.



THE PERCEPTRON: LINEAR SEPARABILITY

- For any linearly separable problem it can be proven using Rosenblatt's (1962) Perceptron Convergence Theorem that the perceptron will find a solution state. However this drawback drastically limits the type of problems the perceptron can solve.

- This limitation was first pointed out in a paper by Minsky and Papert (Minsky & Papert "Perceptrons, An Essay in Computational Geometry") which stated that neural networks would never be able to overcome some basic computational problems, such as the classic **Exclusive OR (XOR)** problem, so there was no possibility of producing intelligent machines using neural networks.

THE PERCEPTRON: LINEAR SEPARABILITY

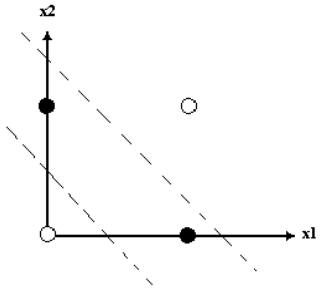

Marvin Minsky


Seymour Papert

- Minsky and Papert wrote the book Perceptrons, which became the foundational work in the analysis of artificial neural networks. This book is the centre of a controversy in the history of AI, as some claim it to have had great importance in driving research away from neural networks in the 1970s, and contribed to the so-called AI winter. Minsky has also written on the possibility that extraterrestrial life may think like humans, permitting communication. Minsky was an adviser on the movie 2001: A Space Odyssey and is referred to in the movie and book.

THE PERCEPTRON: XOR PROBLEM

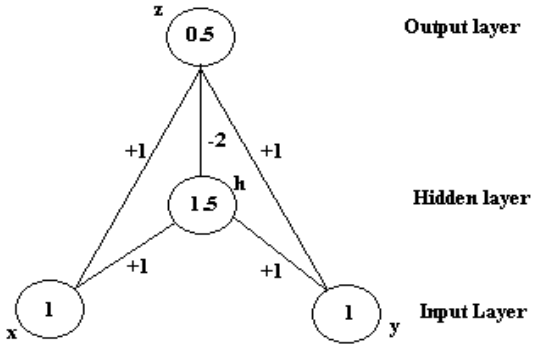
- One of Minsky & Papert's criticisms was not how the perceptron learned but how it represents knowledge. In other words if we could draw an elliptical shape to encircle the two '1' outputs represented by the black dots in the diagram above it would be a much better way of representing the perceptrons knowledge. However perceptrons are unable to model such surfaces.



x2	x1	x1 XOR x2
0	0	0
0	1	1
1	0	1
1	1	0

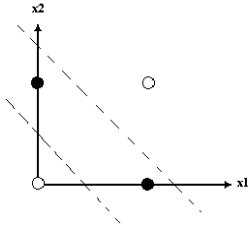
THE PERCEPTRON: XOR PROBLEM

- One approach to solving this knowledge representation problem is to add a hidden node to the perceptron. This node acts as an inhibitory node if the inputs are (0,0) or (1,1). A perceptron capable of solving the XOR problem is shown in the diagram below.



THE PERCEPTRON: XOR PROBLEM

- **Input x = 1 and y = 1**
  - h = (1 x 1) + (1 x 1) = 2 > 1.5 => Fires (output 1)
  - z = (1 x 1) + (1 x 1) + (1 x -2) = 0 > 0.5 => Does not fire (output 0)
- **Input x = 0 and y = 1**
  - h = (0 x 1) + (1 x 1) = 1 > 1.5 => Does not fire (output 0)
  - z = (0 x 1) + (1 x 1) + (0) = 1 > 0.5 => Fires (output 1)



x2	x1	x1 XOR x2
0	0	0
0	1	1
1	0	1
1	1	0

THE PERCEPTRON: CREDIT ASSIGNMENT PROBLEM

- The hidden unit h in the diagram above recognises that both inputs have fired and inhibits the output unit. The use of Multilayer perceptrons solves the knowledge representation problem but introduces serious learning problem.
- The Convergence Theorem does not extend to multilayer perceptrons. The perceptron learning algorithm can correctly adjust weights between inputs but it cannot adjust weights between hidden nodes, this problem is commonly known as the **'Credit Assignment Problem'** .
- In the previous example the inhibitory weight of **1.5** was hand coded, not learned. At the time Minsky & Papert published their paper no one knew how multilayer perceptrons could be made to learn. Despite the identification of this important research problem, actual research in perceptron learning came to a halt in the 1970's.

ADALINE NETWORK

- The ADALINE network developed by Bernard Widrow (1963) is a simple processing element capable of sorting a set of input patterns into two categories. This sorting task may sound like a trivial classification problem but the characteristic that sets the ADALINE apart from traditional methods is its ability to learn through a supervised learning process.

- The ADALINE is trained by repeatedly presenting it a training set composed of input patterns and their desired outputs. Learning occurs as the ADALINE minimizes the number of errors it makes when sorting the patterns into their correct categories. Once trained the ADALINE can categorize new inputs according to the experience it gained through the learning process.

ADALINE NETWORK

- As shown in the diagram on the next slide the ADALINE consists of two layers, an input layer and an output layer. The input layer contains an input node for each input to the network and a bias node. The output layer contains only the ADALINE node which produces the output for the network. Each node in the input layer is connected to the ADALIINE node with a link.

- Each input pattern presented to an ADALINE during training or normal operation should have its components normalised into a predetermined range **(typically -1.0 to 1.0)**. This normalisation prevents one component of the input pattern from dominating and possibly interfering with the networks operation. The desired outputs used in the training set should always be one of two values corresponding to the two categories **(usually -1.0 and 1.0)**.

ADALINE NETWORK

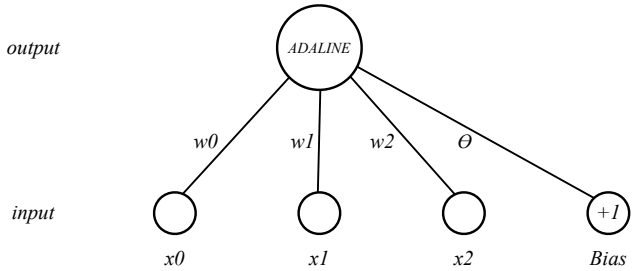
- The ADALINE node produces only two output values, one corresponding to each of the categories the network can discriminate from.

- The output is computed by summing the product of each input node value and its corresponding link value (weight).

- A threshold function is then applied to the weighted sum forcing the output value into one of the two categories. As shown on the next slide the threshold function transforms any positive sum into a value associated with category (1.0) and any negative sum into a value associated with category (-1.0).

ADALINE NETWORK: STEP FUNCTION

ADALINE NETWORK



$$output = f\left(\sum_{i=0}^{N} xi * wi + \theta\right) \qquad f(x) = \begin{cases} 1 \; if \; x > 0 \\ -1 \; otherwise \end{cases}$$

$$output = f\big((x0 * w0) + (x1 * w1) + (x2 * w2) + \theta\big)$$

ADALINE NETWORK: DELTA RULE

- The ADALINE network learns by modifying the weight values associated with the links during the training process. Training is possible due to an approach Widrow and Hoff (1960) devised to minimise the ADALINE's error by adjusting the link values with a learning law commonly referred to as the delta rule.

  Error = Desirder_Output - ADALINE_Output

  New_Weight = Weight + Learning_Rate * Error * Input_Value

- The delta rule uses the error produced by the ADALINE (sorting a pattern into the wrong category) to correct the link values so the correct answer will be produced the next time the input pattern is presented.

ADALINE NETWORK: DELTA RULE

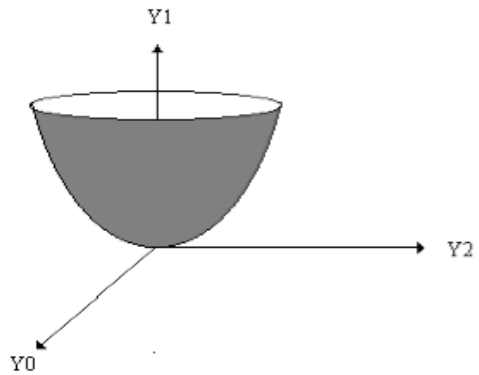
- Since the desired output is either -1 or 1 and the ADALINE's output is either -1 or 1, the error can only be -2 or 2. The delta rule changes the weight values by a percentage called the learning rate (usually around 45%).

- Unfortunately, when the link value is adjusted to give the correct output for the current input pattern, it may cause other input patterns that were producing correct outputs to produce incorrect outputs.

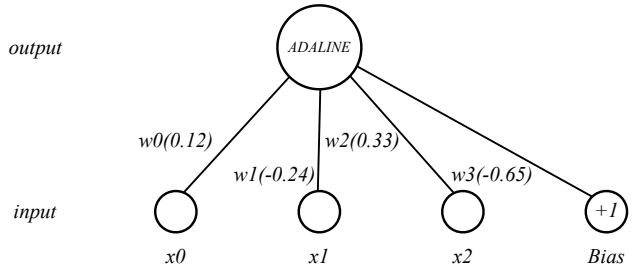
- The training set must be presented many times before all of the input patterns will produce the desired outputs. If the total error for training set is plotted for all possible link values, the error surface will be a paraboliod as shown in the following figure, whose low point is the set of link values that produces the minimum error for each input pattern in the training set.

ADALINE NETWORK: ERROR SURFACE

- The delta rule traverses the error surface downhill until it reaches the minimum point in the paraboloid. Therefore, the initial link values are not important and can be set to a random value (usually between -1.0 and 1.0) before training begins.

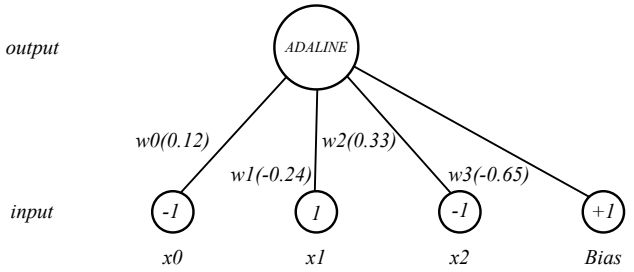


ADALINE EXAMPLE



Pattern No.	Input 1	Input 2	Input 3	Desired Output
1	-1	1	-1	1
2	1	-1	-1	1
3	-1	-1	-1	-1
4	-1	-1	1	1

ADALINE EXAMPLE - FEED FORWARD



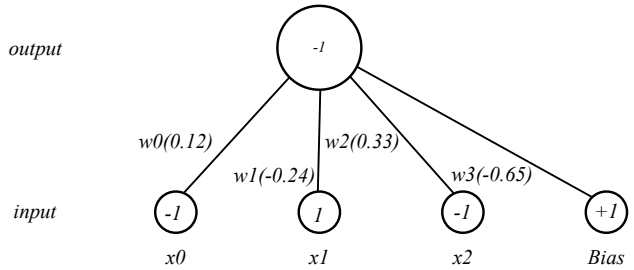
Pattern No.	Input 1	Input 2	Input 3	Desired Output
1	-1	1	-1	1

$$output = f\big((-1 * 0.12) + (1 * -0.24) + (-1 * 0.33) + (1 * -0.65)\big)$$

$$output = f(-1.34)$$

$$output = -1$$

ADALINE EXAMPLE - WEIGHT CHANGE



ERROR = DESIRED – ACTUAL = 1 – (-1)

= **2**

Update each weight using the ERROR, LEARNING RATE and the DELTA RULE:

New_Weight = Weight + Learning_Rate * Error * Input_Value

*w0 = 0.12 + 0.45 * 2 * -1 =* ***-0.78*** *w1 = -0.24 + 0.45 * 2 * 1 =* ***0.66***

*w2 = 0.33 + 0.45 * 2 * -1 =* ***-0.57*** *w3 = -0.65 + 0.45 * 2 * 1 =* ***0.25***