G5AIAI - Introduction to Artificial Intelligence

This course is run at the The University of Nottingham within the School of Computer Science & IT. The course is run by Graham Kendall (EMAIL : gxk@cs.nott.ac.uk)

Neural Networks

Introduction

In this section of the course we are going to consider neural networks. More correctly, we should call them Artificial Neural Networks (ANN) as we not building neural networks from animal tissue. Rather, we are simulating, on a computer, what we understand about neural networks in the brain. But, during this course we will use the term neural network and artificial neural network interchangeably.

You are encouraged to take a look at the reading list for this part of the course.

We start this section of the course by looking at a brief history of the work done in the field of neural networks.

Next we look at how a real brain operates (or as much as we know about how a real brain operates). This will provide us with a model we can use in implementing a neural network.

Following this we will look at how we can solve simple algebraic problems using a neural network. In doing so we will discover the limitations of such a model.

As with other parts of the course I have used AIMA (Russell, 1995), where possible. In addition, some of the material is also based on (Fausett, 1994), which is a good introductory text to neural networks as is (Aleksander, 1995). You might also take a look at (Davalo, 1991) and (Callan, 1998).
In addition, many of the seminal papers (for example, McCulloch/Pitts, Minksy/Papert, Widrow, Hebb etc.) can be found in (Anderson, 1988).

Research History

McCulloch & Pitts (McCulloch, 1943) are generally recognised as being the designers of the first neural network. They recognised that combining many simple processing units together could lead to an overall increase in computational power.
Many of the ideas they suggested are still in use today. For example, the idea that a neuron has a threshold level and once that level is reached the neuron fires is still the fundamental way in which artificial neural networks operate.
The McCulloch and Pitts network had a fixed set of weights and it was Hebb (Hebb, 1949) who developed the first learning rule. His premise was that if two neurons were active at the same time then the strength between them should be increased.

In the fifties and throughout the sixties many researchers worked on the ***perceptron*** (Block, 1962, Minsky & Papert, 1988 (originally published in 1969) and Rosenblatt, 1958, 1959 and 1962).
This neural network model can be proved to converge to the correct weights, if there are weights that will solve the problem.
The learning algorithm (i.e. weight adjustment) used in the perceptron is more powerful than the learning rules used by Hebb.

The perceptron caused great excitement at the time as it was thought it was the path to producing programs that could think. But in 1969 (Minsky & Papert, 1969) it was shown that the perceptron had severe limitations which meant that it could not learn certain types of functions (i.e. those which are not linearly separable).

Due to Minsky and Papert's proof that the perceptron could not learn certain type of (important) functions, research into neural networks went into decline throughout the 1970's.

It was not until the mid 80's that two people (Parker, 1985) and (LeCun, 1986) independently discovered a learning

algorithm for multi-layer networks called backpropogation that could solve problems that were not linearly separable. In fact, the process had been discovered in (Werbos, 1974) and was similar to another algorithm presented by (Bryson & Ho, 1969) but it took until the mid eighties to make the link to neural networks.

The Brain

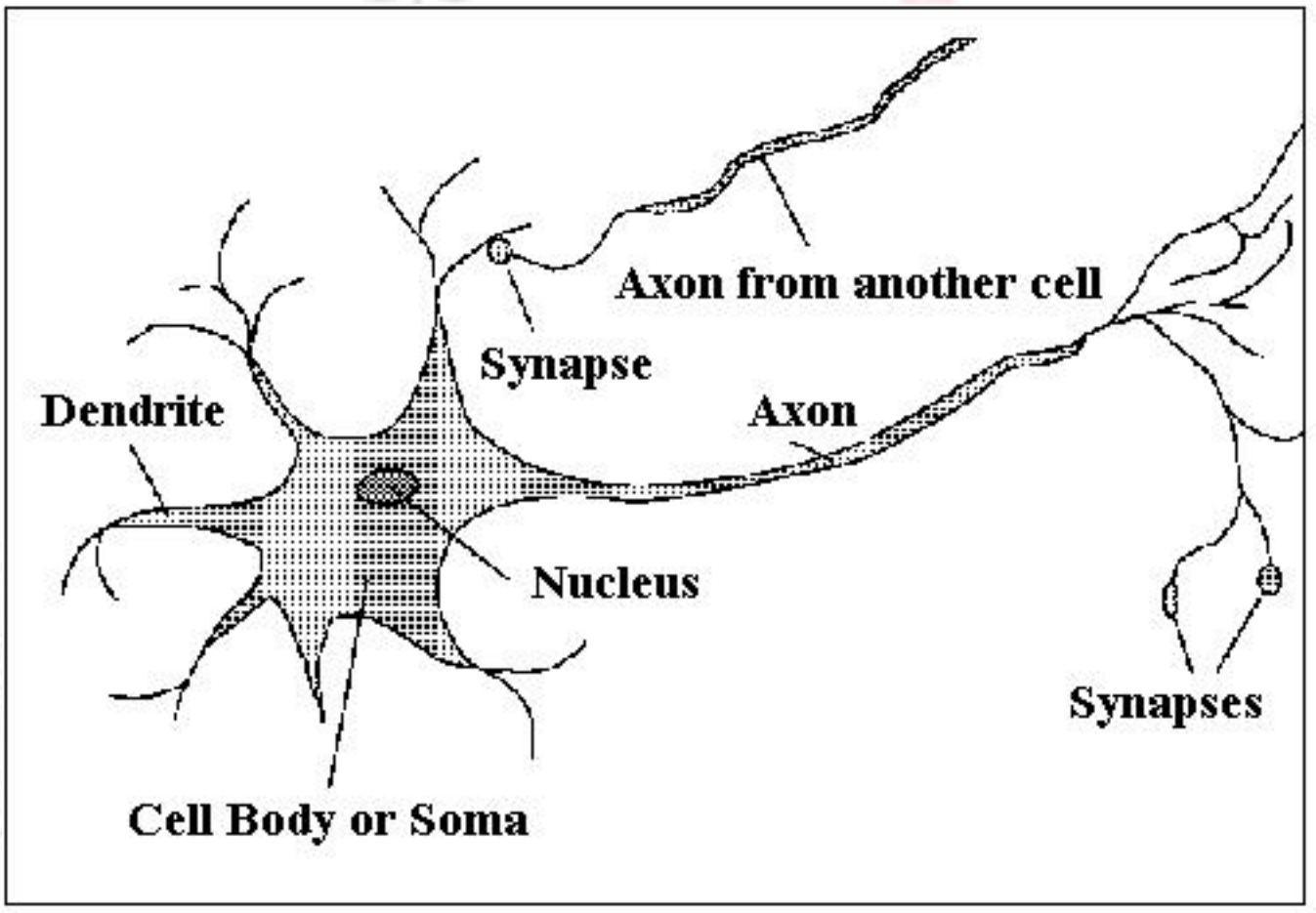
We still do not know exactly how the brain works. For example, we are born with about 100 billion neurons in our brain. Many die as we progress through life, and are not replaced, yet we continue to learn.

Although we do not know exactly how the brain works, we do know certain things about it.
We know it is resilient to a certain amount of damage (in addition to the continual loss we suffer as we get older). There have been reports of objects being passed (if passed is the right word) all the way through the brain with only slight impairment to the persons mental capability.
We also know what certain parts of the brain do. We know, for example, that much information processing goes on in the cerebral cortex, which is the outer layer of the brain.

From a computational point of view we also know that the fundamental processing unit of the brain is a neuron.

- A neuron consists of a cell body, or soma, that contains a nucleus.
- Each neuron has a number of dendrites which receive connections from other neurons.
- Neurons also have an axon which goes out from the neuron and eventually splits into a number of strands to make a connection to other neurons.
- The point at which neurons join other neurons is called a synapse.
- A neuron may connect to as many as 100,000 other neurons.

A simplified view of a neuron is shown in the diagram below.



Signals move from neuron to neuron via electrochemical reactions. The synapses release a chemical transmitter which enters the dendrite. This raises or lowers the electrical potential of the cell body.
The soma sums the inputs it receives and once a threshold level is reached an electrical impulse is sent down the axon (often known as firing).
These impulses eventually reach synapses and the cycle continues.Synapses which raise the potential within a cell body are called excitatory. Synapses which lower the potential are called inhibitory.It has been found that synapses exhibit plasticity. This means that long-term changes in the strengths of the connections can be formed depending on the firing patterns of other neurons. This is thought to be the basis for learning in our brains.

The First Artificial Neuron

Much of this section is taken from (Fausett, 1994).
As mentioned in the research history McCulloch and Pitts (1943) produced the first neural network, which was based on their artificial neuron. Although this work was developed in the early forties, many of the principles can still be seen in the neural networks of today.

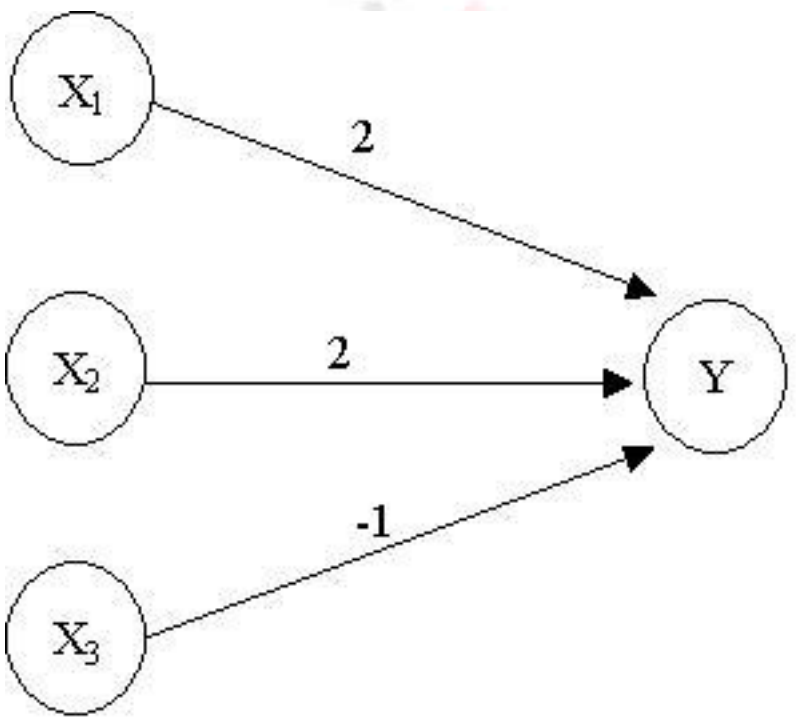
We can make the following statements about a McCulloch-Pitts network

- The activation of a neuron is binary. That is, the neuron either fires (activation of one) or does not fire (activation of zero).
  For the network shown below the activation function for unit Y is

$f(y\_in) = 1$, if $y\_in >= t$ else 0

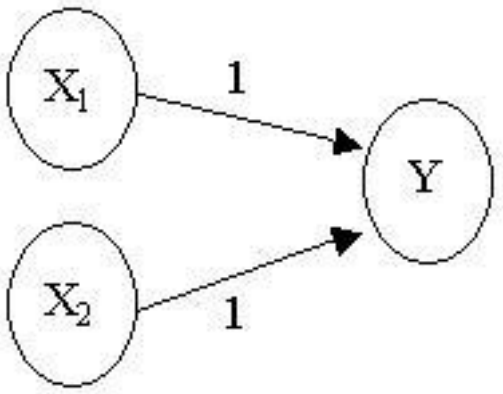
where y_in is the total input signal received
t is the threshold for Y.

- Neurons is a McCulloch-Pitts network are connected by directed, weighted paths.
- If the weight on a path is positive the path is excitatory, otherwise it is inhibitory.
- All excitatory connections into a particular neuron have the same weight, although different weighted connections can be input to different neurons.
- Each neuron has a fixed threshold. If the net input into the neuron is greater than the threshold, the neuron fires.
- The threshold is set such that any non-zero inhibitory input will prevent the neuron from firing.
- It takes one time step for a signal to pass over one connection.

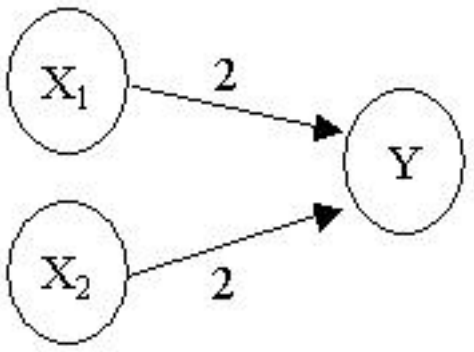


A sample McCulloch-Pitts network is shown above and some of the statements can be observed.
In particular, note that the threshold for Y has equal 4 as this is the only value that allows it to fire, taking into account that a neuron cannot fire if it receives a nonzero inhibitory input.

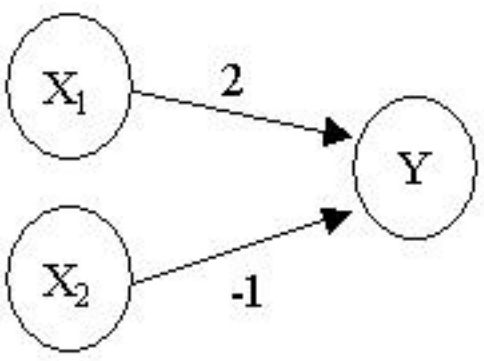
Using the McCulloch-Pitts model we can model logic functions. Below we show and describe the architecture for four logic functions (the truth tables for each function is also shown)



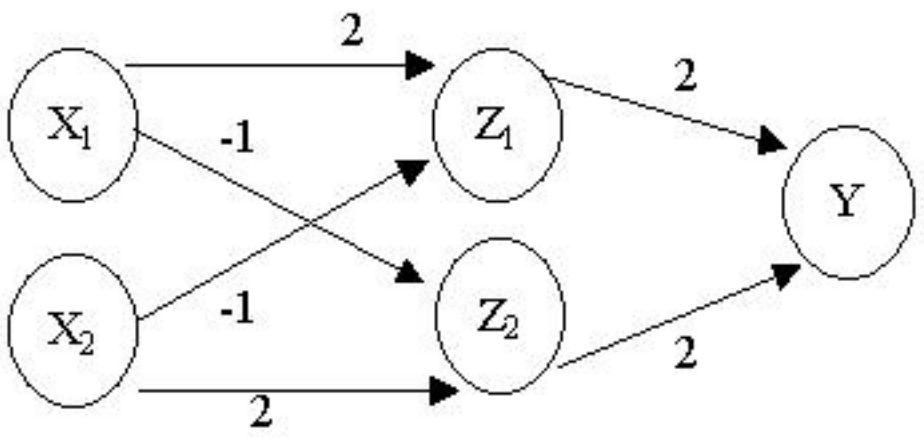
AND Function



OR Function



AND NOT Function



XOR Function

AND		
X_1	X_2	Y
1	1	1
1	0	0
0	1	0
0	0	0

OR		
X_1	X_2	Y
1	1	1
1	0	1
0	1	1
0	0	0

AND NOT		
X_1	X_2	Y
1	1	0
1	0	1
0	1	0
0	0	0

XOR		

X_1	X_2	Y
1	1	0
1	0	1
0	1	1
0	0	0

AND Function

As both inputs (X_1 and X_2) are connected to the same neuron the connections must be the same, in this case 1. To model the AND function the threshold on Y is set to 2.

OR Function

This is almost identical to the AND function except the connections are set to 2 and the threshold on Y is also set to 2.

AND NOT Function

Although the truth table for the AND NOT function is shown above it deserves just a small explanation as it is not often seen in the textbooks. The function is not symmetric in that an input of 1,0 is treated differently to an input of 0,1. As you can see from the truth table the only time true (value of one) is returned is when the first input is true and the second input is false.
Again, the threshold on Y is set to 2 and if you apply each of the inputs to the AND NOT network you will find that we have modeled X_1 AND NOT X_2.

XOR Function

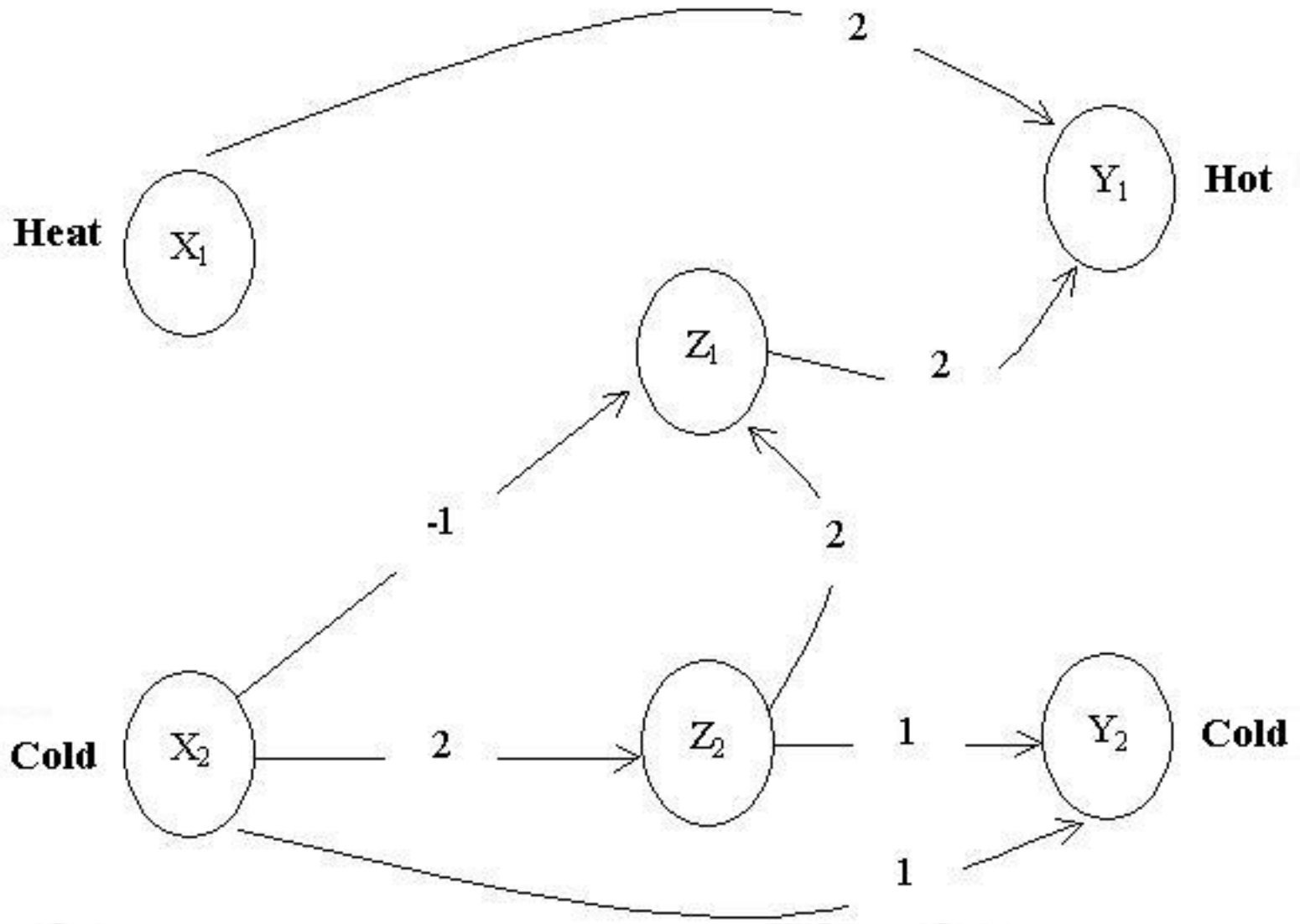
XOR can be modeled using AND NOT and OR;

$$X_1 \text{ XOR } X_2 = (X_1 \text{ AND NOT } X_2) \text{ OR } (X_2 \text{ AND NOT } X_1)$$

(To prove it draw the truth table)
This explains the network shown above. The first layer performs the two AND NOT's and the second layer performs the OR. Both Z neurons and the Y neuron have a threshold of 2.

As a final example of a McCulloch-Pitts network we will consider how to model the phenomenon that if you touch something very cold you initially perceive heat. Only after you have left your hand on the cold source for a while do you perceive cold. This example (from Fausett, 1994) is an elaboration of one originally presented by (McCulloch and Pitts, 1943).
To model this we will assume that time is discrete. If cold is applied for one time step then heat will be perceived. If a cold stimulus is applied for two time steps then cold will be perceived. If heat is applied then we should perceive heat.

Take a look at this figure. Each neuron has a threshold of 2. This, as we shall see, allows us to model this phenomenon.



First though, remember that time is discrete, so it takes time for the stimulus (applied at X_1 and X_2) to make its way to Y_1 and Y_2 where we perceive either heat or cold.
Therefore, at t(0), we apply a stimulus to X_1 and X_2.
At t(1) we can update Z_1, Z_2 and Y_1.
At t(2) we can perceive a stimulus at Y_2.
At t(2+n) the network is fully functional.

Before we see if the network performs as we hope let's consider what we are trying to do.

Input to the system will be (1,0) or (0,1), which represents hot and cold respectively.
We want the system to perceive cold if a cold stimulus is applied for two time steps. That is

$$Y_2(t) = X_2(t - 2) \textbf{ AND } X_2(t - 1) \quad (\textit{formula 1})$$

This truth table (i.e. the truth table for AND) shows that this model is correct

$X_2(t - 2)$	$X_2(t - 1)$	$Y_2(t)$
1	1	1
1	0	0
0	1	0
0	0	0

We want the system to perceive heat if either a hot stimulus is applied or a cold stimulus is applied (for one time step) and then removed. We can express this as follows

$$Y_1(t) = [\ X_1(t - 1)\ ] \textbf{ OR } [\ X_2(t - 3) \textbf{ AND NOT } X_2(t - 2)\ ] \quad (\textit{formula 2})$$

This truth table shows that it gives us the required result

X2(t-3)	X2(t-2)	AND NOT	X1(t-1)	OR

1	1	0	1	1
1	0	1	1	1
0	1	0	1	1
0	0	0	1	1
1	1	0	0	0
1	**0**	**1**	**0**	**1**
0	1	0	0	0
0	0	0	0	0

The way to read this table is to firstly consider the first three columns. This gives the result (in the AND NOT column) of the AND NOT expression in the formula. This result is then OR'ed with the $X_1(t\text{-}1)$ to give the final column in the table, which is the result of the entire expression.

You can see that if we apply heat (i.e. $X_1(t\text{-}1)$) then we always perceive heat as the output. If we apply cold for one time step (i.e. the row in red) we also perceive heat as the output.

So, if we are convinced that we have the correct logical statements to represent the hot/cold problem, we now need to convince ourselves that the network above represents the logical statements.

The figure of the network shows that

$$Y_1(t) = X_1(t - 1) \textbf{ OR } Z_1(t - 1) \quad (\textit{formula 3})$$

(compare this to the OR network we developed earlier).

Now Z_1 neuron. This is how it is formed

$$Z_1(t - 1) = Z_2(t - 2) \textbf{ AND NOT } X_2(t - 2) \quad (\textit{formula 4})$$

(again, compare this to the AND NOT network above).

Now Z_2, this is simply

$$Z_2(t - 2) = X_2(t - 3) \quad (\textit{formula 5})$$

If we take formula 3, and substitute in formula 4 and 5 we end up with

$$Y_1(t) = [\, X_1(t - 1) \,] \textbf{ OR } [\, X_2(t - 3) \textbf{ AND NOT } X_2(t - 2) \,] \quad (\textit{formula 6})$$

which is the same as formula 2, showing that our network (*Y1* anyway) works correctly (as we have proved formula 2 works using a full analysis by using the truth table).

We can perform a similar analysis for Y_2, and show that Y_2 in the network acts in the way we developed above (formula 1). You should do this to convince yourself that this is correct.

If you still don't believe it, there is a spreadsheet available from the course web site that implements this network so that you can see that it works as we expect. Note, that this spreadsheet (worsheet really, if we are going to use Microsoft terminology) has two spreadsheets within it. One called "ReadMe" gives some general information. The other, "HotCold", contains the implementation.

Modelling a Neuron

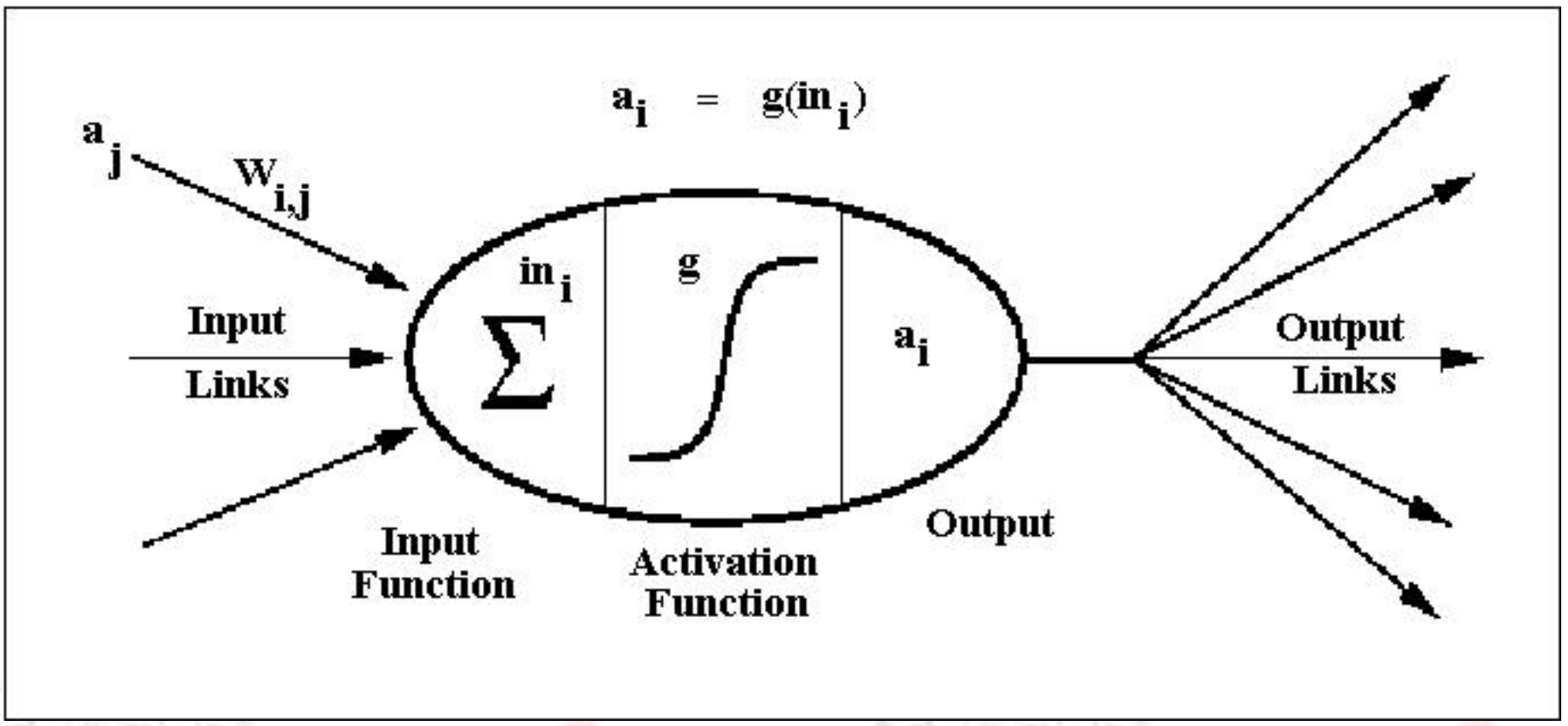
To model the brain we need to model a neuron. Each neuron performs a simple computation. It receives signals from its input links and it uses these values to compute the activation level (or output) for the neuron. This value is passed to other neurons via its output links.

The input value received of a neuron is calculated by summing the weighted input values from its input links. That is

$$in_i = \sum_j W_{j,i} a_j$$

An activation function takes the neuron input value and produces a value which becomes the output value of the neuron. This value is passed to other neurons in the network.

This is summarised in this diagram and the notes below.



a_j : Activation value of unit j

$w_{j,i}$: Weight on the link from unit j to unit i

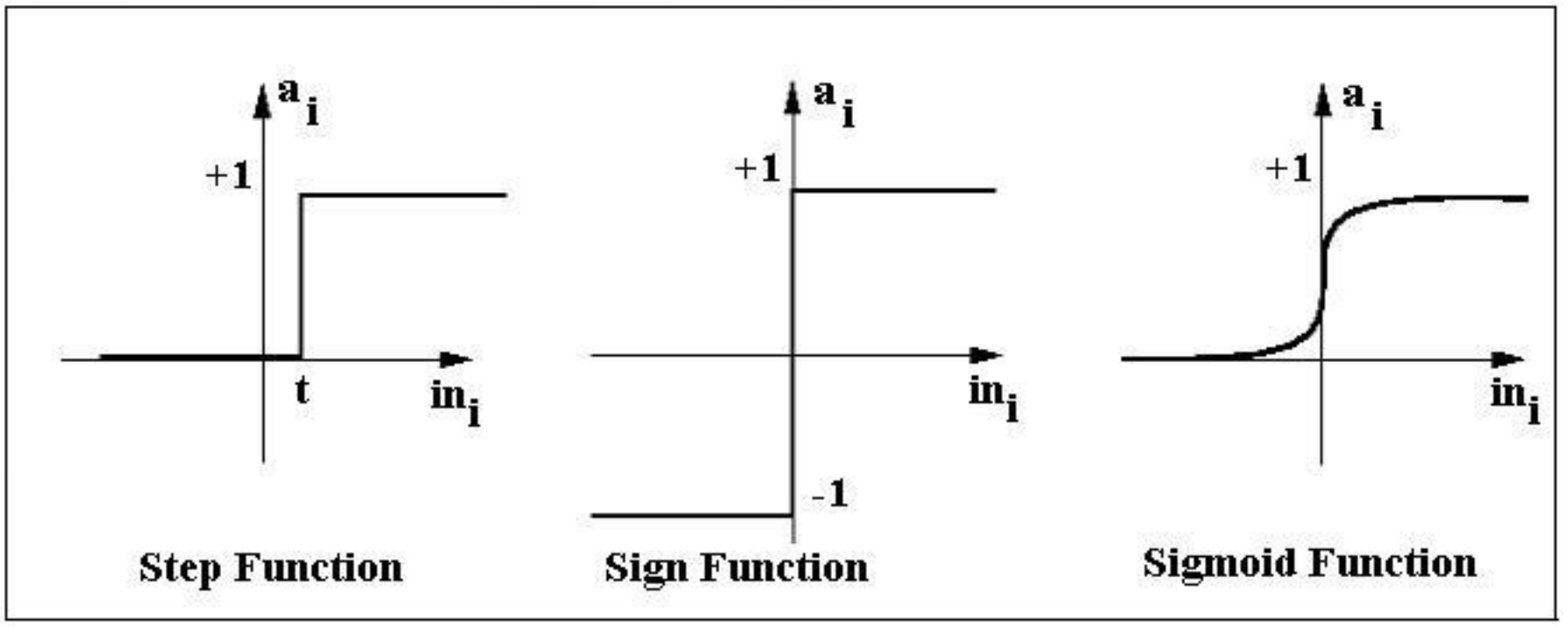
in_i : Weighted sum of inputs to unit i

a_i : Activation value of unit i (also known as the output value)

g : Activation function

Or, in English.
A neuron is connected to other neurons via its input and output links. Each incoming neuron has an activation value and each connection has a weight associated with it.
The neuron sums the incoming weighted values and this value is input to an activation function. The output of the activation function is the output from the neuron.

Some common activation functions are shown below.



These functions can be defined as follows.

$Step_t(x)$ = 1 if x >= t, else 0
Sign(x) = +1 if x >= 0, else -1
Sigmoid(x) = $1/(1+e^{-x})$

On occasions an identify function is also used (i.e. where the input to the neuron becomes the output). This function is normally used in the input layer where the inputs to the neural network are passed into the network unchanged.

Some Simple Networks

We can use what we have learnt above to demonstrate a simple neural network which acts as a logic gate. The diagram below is modelling the following truth tables

AND

$Input_1$	$Input_2$	Output
0	0	0
0	1	0
1	0	0
1	1	1

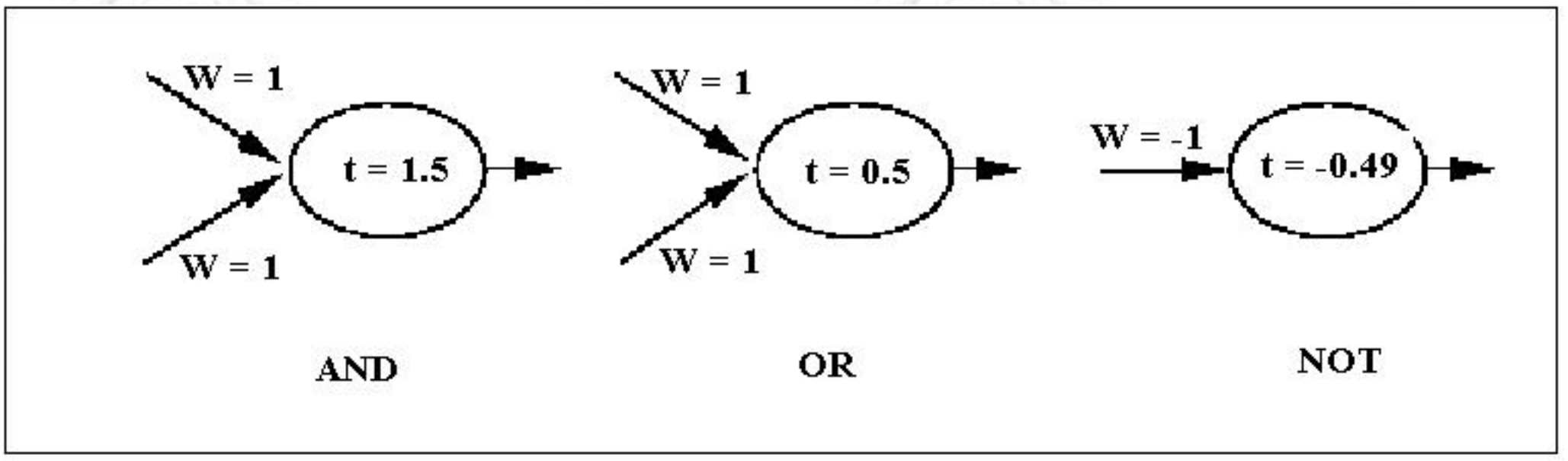
OR

$Input_1$	$Input_2$	Output
0	0	0
0	1	1
1	0	1
1	1	1

NOT

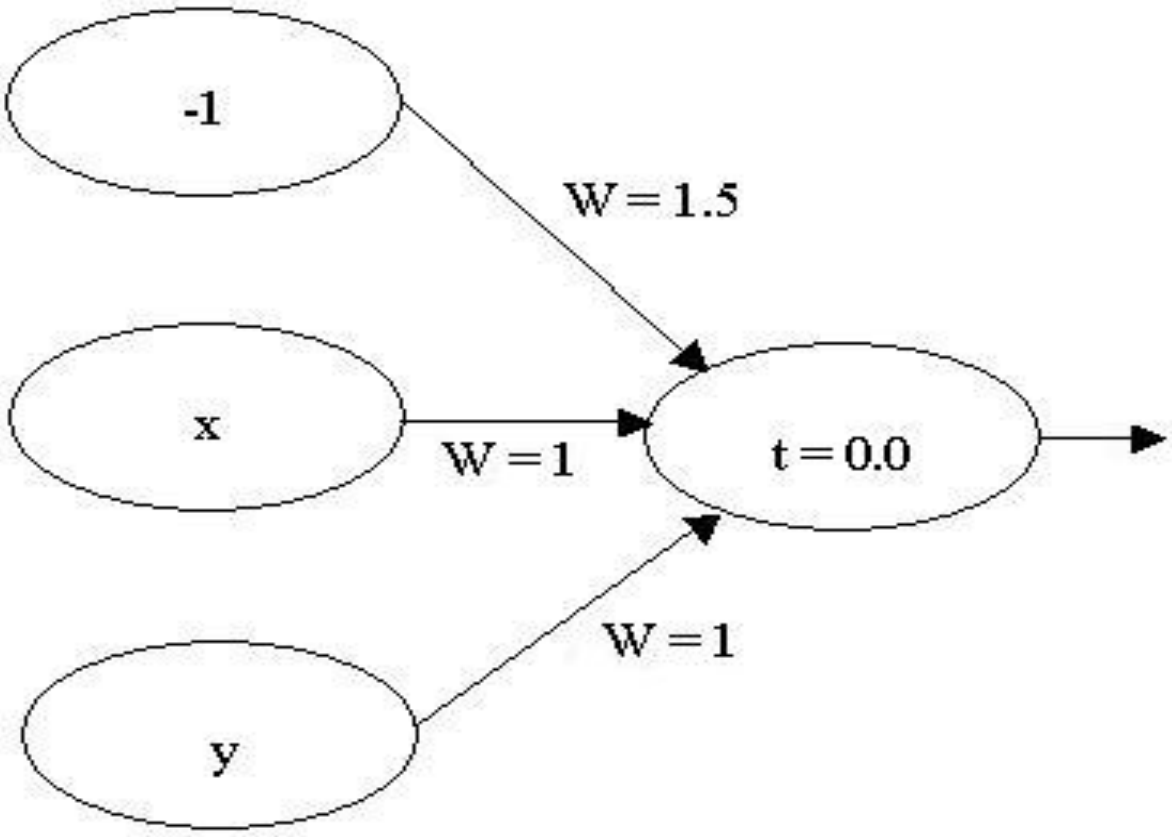
Input	Output
0	1

1	0



In these networks we are using the step activation function.
You should convince yourself that these networks produce the correct output for all the allowed inputs.

You will notice that each neuron has a different threshold. From a computational viewpoint it would be easier if all the neurons had the same threshold value and the actual threshold was somehow modelled by the weights.
In fact, it is possible to do exactly this. Consider this network



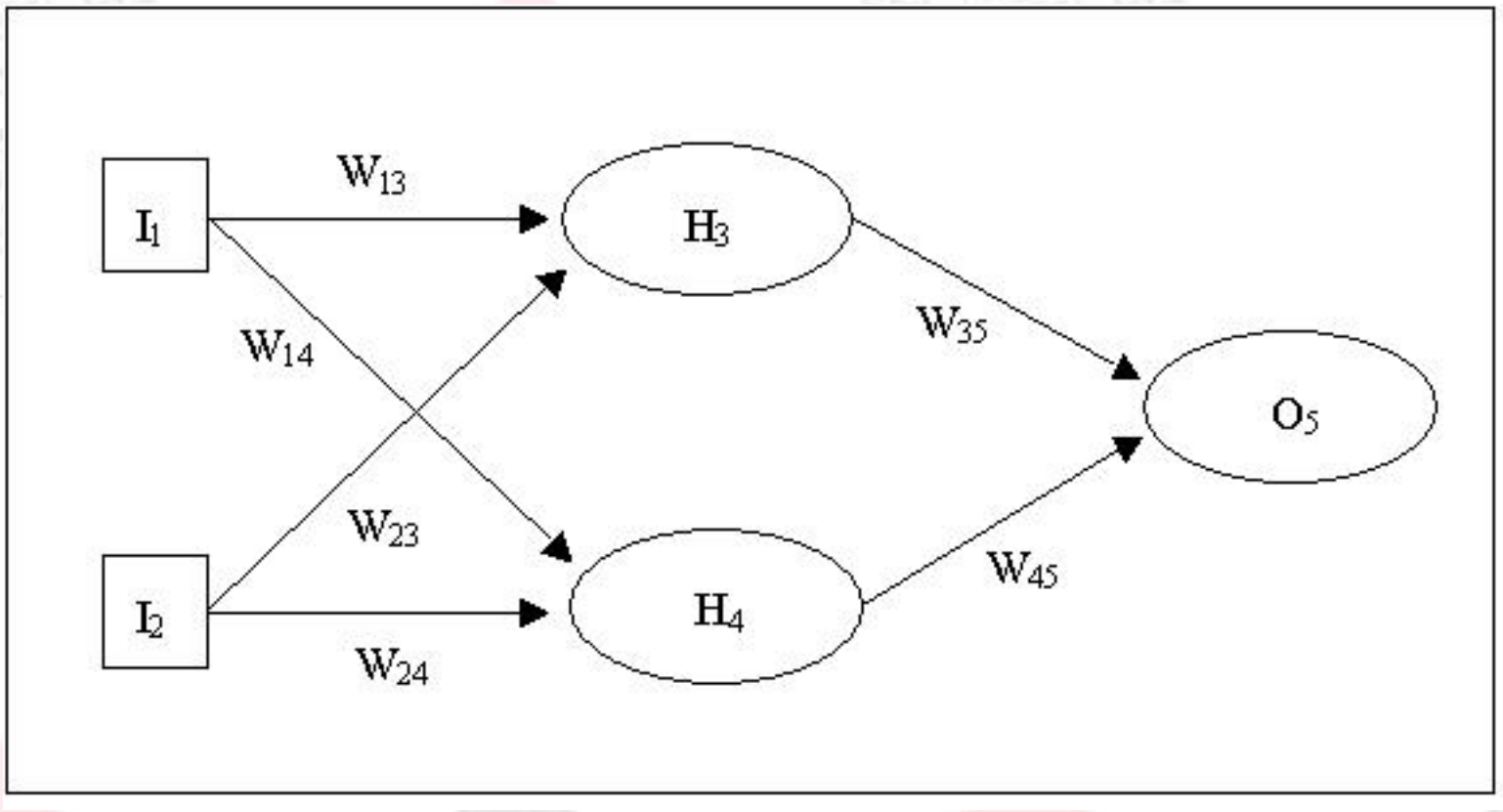
It is the same as the AND network in the diagram above except that there is an extra input neuron whose activation is always set to -1. The threshold of the neuron is represented by the weight that links this extra neuron to the output neuron.
This means the threshold of the neuron can be set to zero.

You might like to work through this network using the four possible combinations of x and y and convince yourself that the network operates correctly.

This advantage of this method is that we can always set the threshold for every neuron to zero and from a computational point of view, when "training" the network, we only have to update weights and not both thresholds and weights.

Types of Network

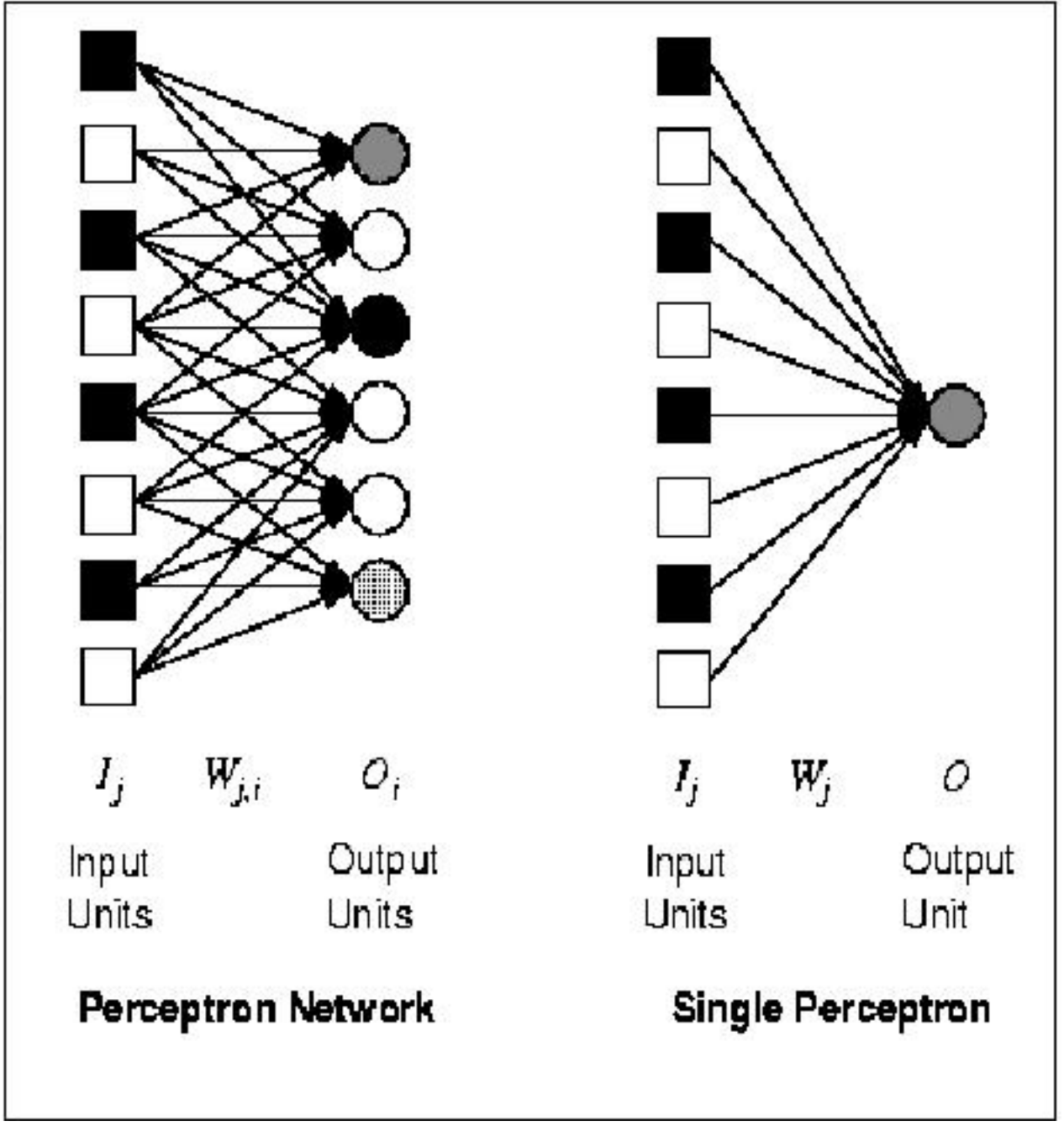
The simple networks we have considered above only have input neurons and output neurons. It is considered a one layer network (the input neurons are not normally considered to form a layer as they are just a means of getting data into the network).
Also, in the networks we have considered, the data only travels in one direction (from the input neurons to the output neurons). In this respect it is known as a feed-forward network.
Therefore, we have been looking at one-layer, feed-forward networks.
There are many other types of network. An example of a two-layer, feed-forward network is shown below.



The perceptron

The name perceptron is now used as a synonym for single-layer, feed-forward networks. They were first studied in the 1950's and although other network architectures were known about the perceptron was the only network that was known to be capable of learning and thus most of the research at that time concentrated on perceptrons.

The diagram below shows example of perceptrons.



You will see, in the left hand network that a single weight only affects one of the outputs. This means we can make our study of perceptrons easier by only considering networks with a single output (i.e. similar to the network shown on the right hand side of the diagram).

As we only have one output we can make our notation a little simpler. Therefore the output neuron is denoted by O and the weight from input neuron j is denoted by Wj.
Therefore, the activation function becomes

$$O = Step_0 \sum_j WjIj$$

(Note, we are assuming the use of additional weight to act as a threshold so that we can use a $step_0$ function, rather than $step_t$).

What can perceptrons represent?

We have already seen that perceptrons can represent the AND, OR and NOT logic functions. But does it follow that a perceptron (a single-layer, feed-forward network) can represent any boolean function?
Unfortunately, this is not the case.

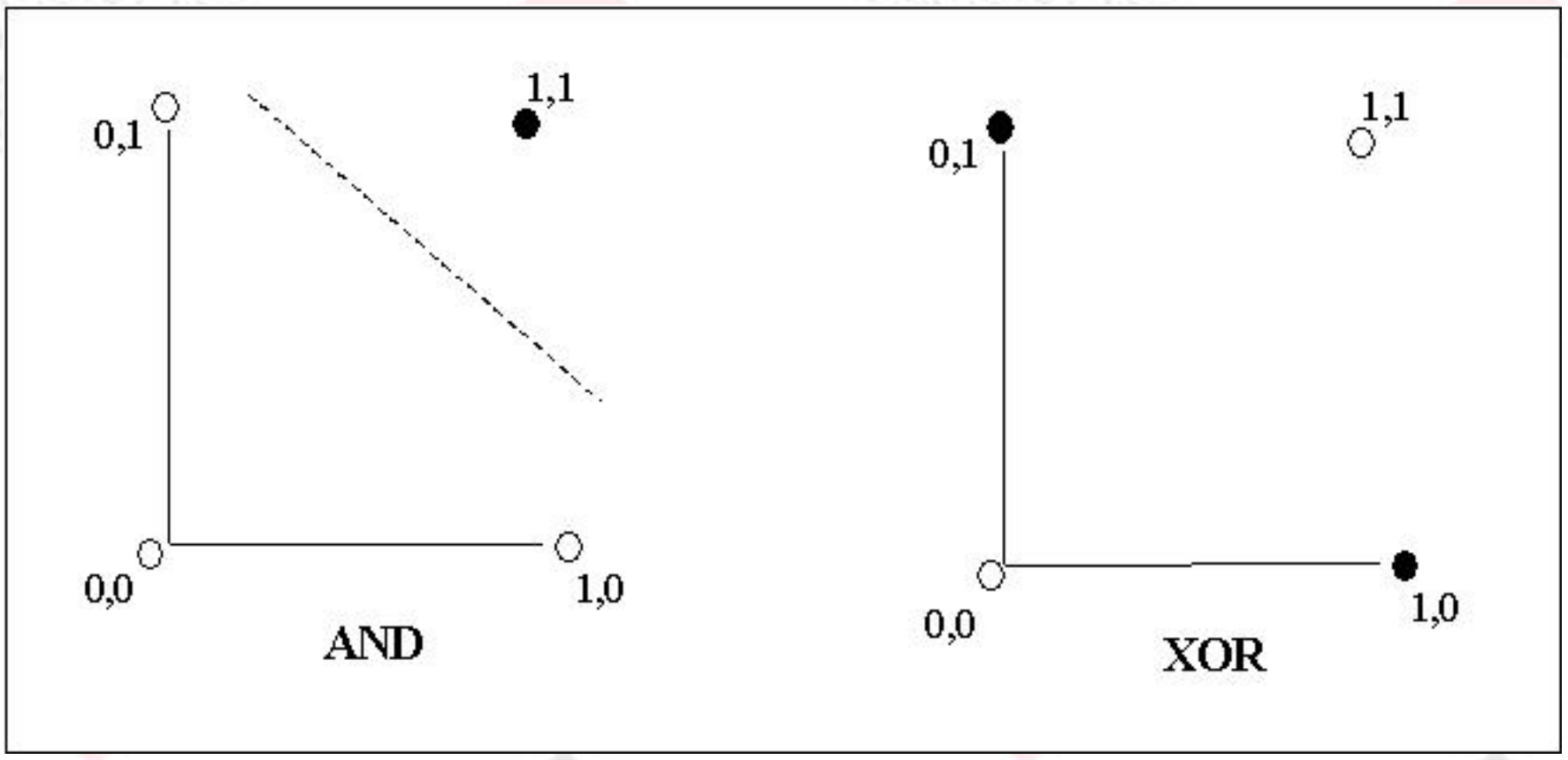
To see why, consider these two truth tables

AND		
$Input_1$	**$Input_2$**	**Output**

0	0	0
0	1	0
1	0	0
1	1	1

XOR		
$Input_1$	**$Input_2$**	**Output**
0	0	0
0	1	1
1	0	1
1	1	0

We can represent these two truth tables graphically. Like this



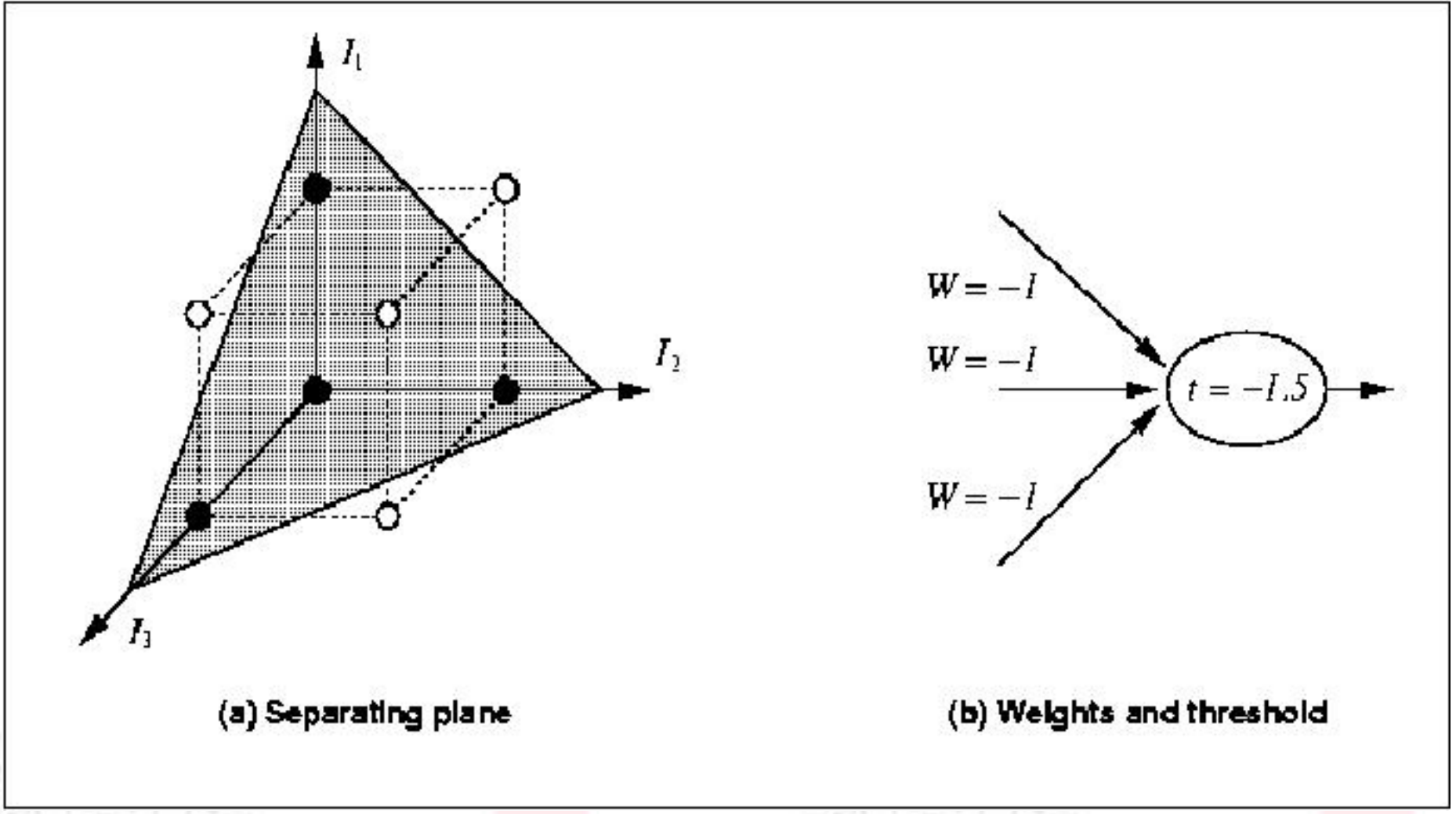
(where a filled circle represents an output of one and a hollow circle represents an output of zero).

If you look at the AND graph you will see that we can divide the one's from the zero's with a line. This is not possible with XOR.

Functions such as AND are called ***linearly separable***.

It was the proof by Minsky & Papert in 1969 that perceptrons could only learn linearly separable functions that led to decline in neural network research until the mid 1980's when it was proved that other network architectures could learn these type of functions.

Although, only being able to learn linearly separable functions is a major disadvantage of the perceptron it is still worth studying as it is relatively simple and can help provide a framework for other architectures.
It should however, be realised that perceptrons are not only limited to two inputs (e.g. the AND function). We can have *n* inputs which gives us an *n*-dimension problem.
When *n*=3 we can still visualise the linear separability of the problem (see diagram below).



But once *n* is greater than three we find it difficult to visualise the problem.

Learning Lineraly Separable Functions

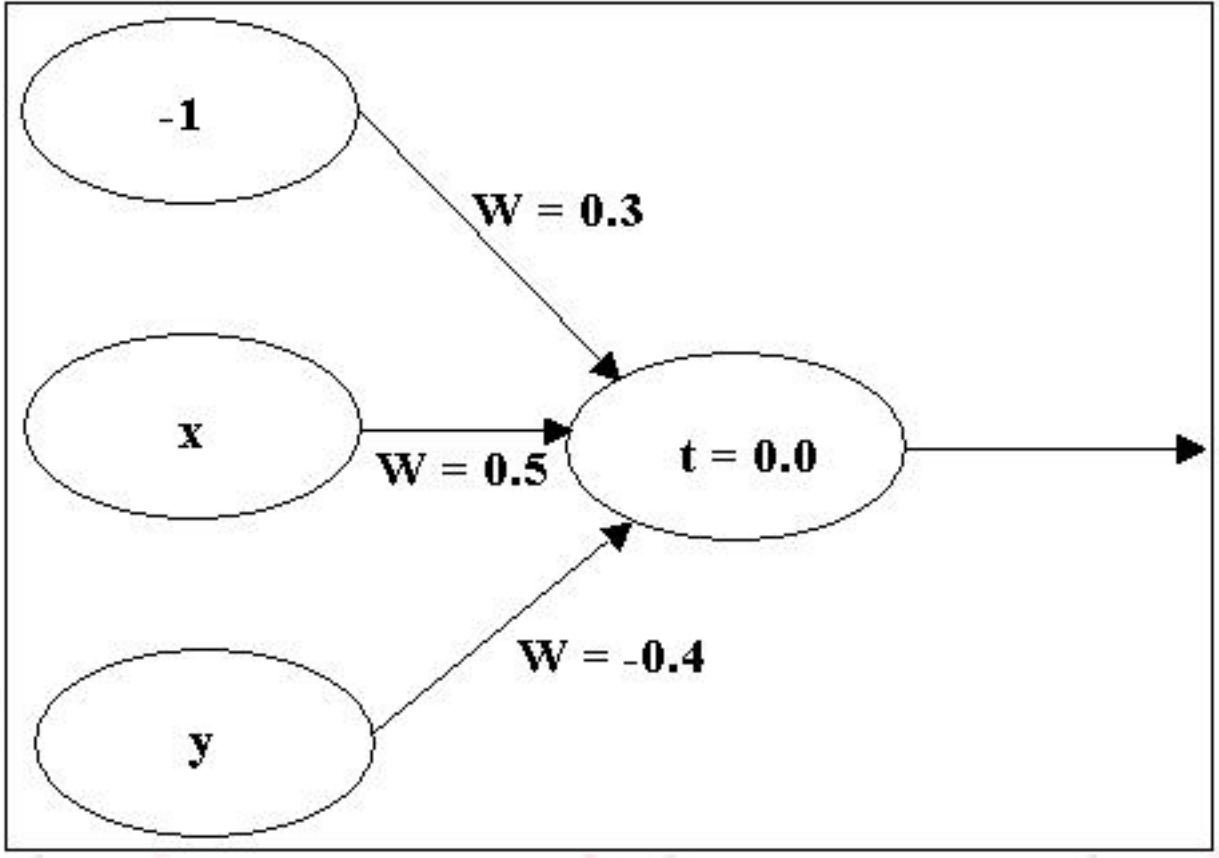
With a function such as AND (with only two inputs) we can easily decide what weights to use to give us the required output from the neuron. But with more complex functions (i.e. those with more than two inputs it may not be so easy to decide on the correct weights).

Therefore, we would like our neural network to "learn" so that it can come up with its own set of weights.

We will consider this aspect of neural networks for a simple function (i.e. one with two inputs). We could obviously scale up the problem to accommodate more complex problems, providing the problems are linearly separable.

Consider this truth table (AND) and the neuron that we ***hope*** will represent it.

AND		
$Input_1$	**$Input_2$**	**Output**
0	0	0
0	1	0
1	0	0
1	1	1

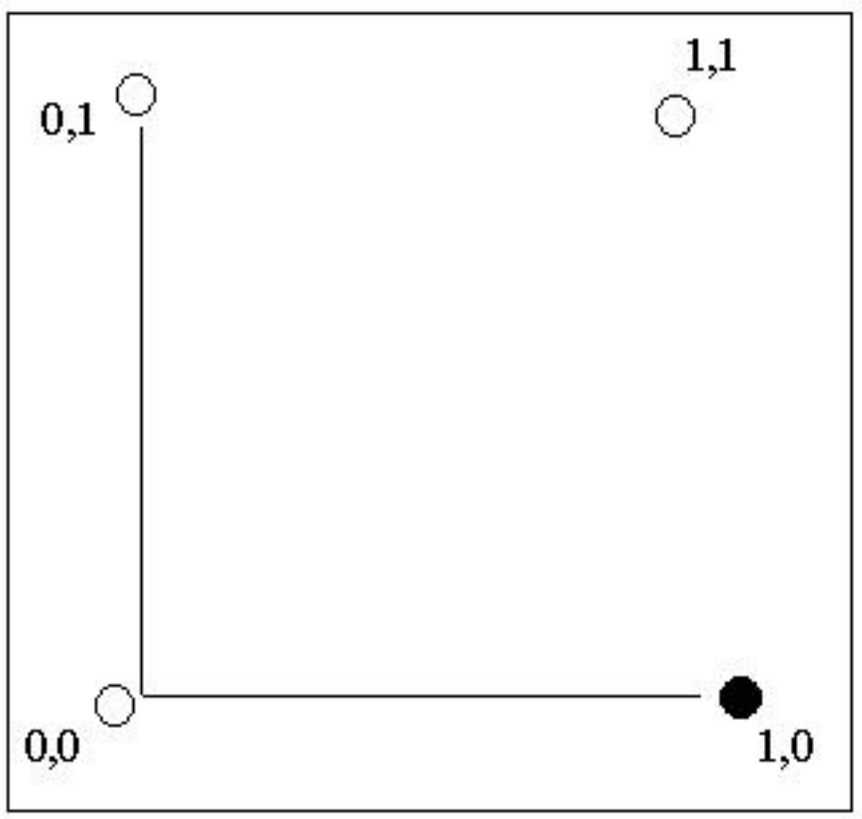


In fact, all we have done is set the weights to random values between -0.5 and 0.5

By applying the activation function for each of the four possible inputs to this neuron (try it) it actually gives us the following truth table

???		
$Input_1$	**$Input_2$**	**Output**
0	0	0
0	1	0
1	0	1
1	1	0

Which can be graphically represented as follows



The network, obviously, does not represent the AND function so we need to adjust the weights so that it "learns" the function correctly.

The algorithm to do this follows, but first some terminology.

Epoch	:	An epoch is the presentation of the entire training set to the neural network. In the case of the AND function an epoch consists of four sets of inputs being presented to the network (i.e. [0,0], [0,1], [1,0], [1,1]).
Training Value, T	:	When we are training a network we not only present it with the input but also with a value that we require the network to produce. For example, if we present the network with [1,1] for the AND function the training value will be 1.
Error, Err	:	The error value is the amount by which the value output by the network differs from the training value. For example, if we required the network to output 0 and it output a 1, then Err = -1. Output from Neuron, O : The output value from the neuron
I_j	:	Inputs being presented to the neuron
W_j	:	Weight from input neuron (Ij) to the output neuron
LR	:	The learning rate. This dictates how quickly the network converges. It is set by a matter of experimentation. It is typically 0.1.

The Perceptron Training Algorithm

```
While epoch produces an error
     Present network with next inputs from epoch
     Err = T - O
     If Err <> 0 then
                Note : If the error is positive we need to increase O.
                       If the error is negative we need to decrease O.
                       Each input contributes WjIj to the total input
                       so if Ij is positive, an increase in Wj will
                       increase O. If Ij is negative an increase in Wj
                       will decrease O).
                       This can be achieved with the following
          Wj = Wj + LR * Ij * Err
                Note : This is often called the delta learning rule.

     End If
End While
```

Perceptron Learning - An Example

Let's take a look at an example. The initial weight values are 0.3, 0.5, and -0.4 (taken from the above example) and we are trying to learn the AND function. You can follow this discussion using the perceptron spreadhseet, which implements a simple neural network and allows us to learn functions such as AND.

If we present the network with the first training pair ([0,0]), from the first epoch, nothing will happen to the weights (due to multiplying by zero).
The next training pair ([0,1]) will result in the network producing zero (by virtue of the $step_0$ function). As zero is the required output there is no error so training continues.
The next training pair ([1,0]) produces an output of one. The required output is 0. Therefore the error is -1. This means we have to adjust the weights.

This is done as follows (assuming LR = 0.1)

$W_0 = 0.3 + 0.1 * -1 * -1 = 0.4$
$W_1 = 0.5 + 0.1 * 1 * -1 = 0.4$
$W_2 = -0.4 + 0.1 * 0 * -1 = -.04$

Therefore, the new weights are 0.4, 0.4, -0.4.

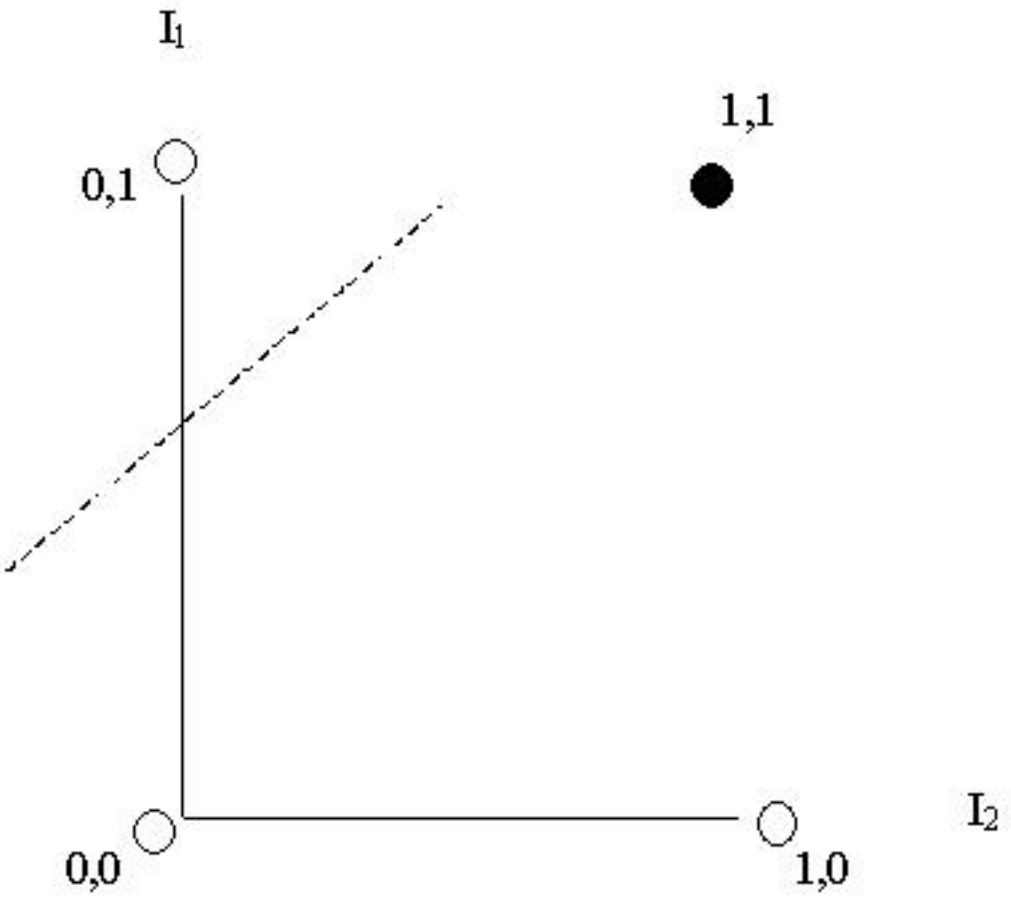
Finally we apply the input [1,1] to the network. This also produces an error and the new weight values will be 0.3, 0.5 and -0.3.

As this presentation of the epoch produced an error (two in fact) we need to continue the training and present the network with another epoch.

Training continues until an epoch is presented that does not produce an error.

If we consider the state of the network at the and of the first epoch (weights = 0.3, 0.5, -0.3) we know the weights are wrong as the epoch produced an error (in fact, the weights *may* be correct at the end of the epoch but we need to present another epoch to show this). We can also produce a graph that shows the current state of the network.

We are trying to achieve the AND function which can be represented as follows



The current "linear separability" line can be drawn by using the weights to draw a line on the graph. Two points on the I_1 and I_2 axis can be found as follows

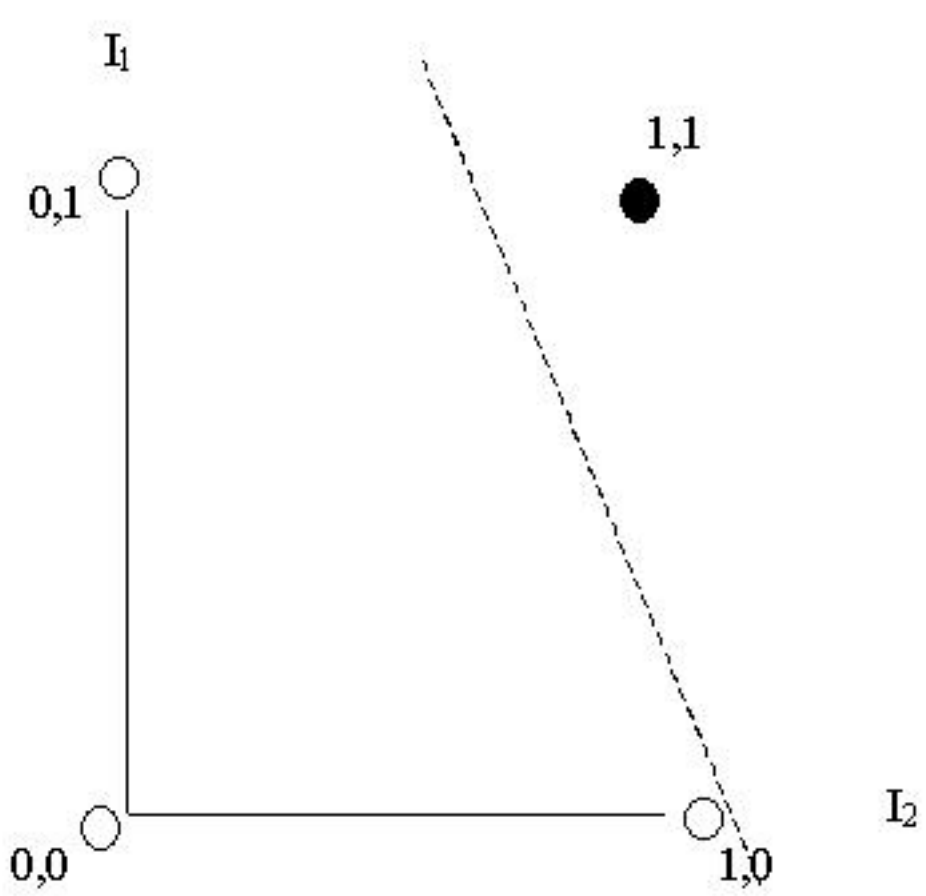
I_1 point = W_0/W_1
I_2 point = W_0/W_2

Note : As discussed above, W_0 actually represents the threshold. Therefore, we are dividing the threshold by the weights.

That is, $I_1 = 0.6$ and $I_2 = -1$

This line is shown (*roughly* in the right position) on the above graph. It is clearly not in the correct place for the AND function as the line does not divide the one's and the zero's correctly (i.e separate the filled and empty circles).

If we continue with the training (and you should try this) until we get no errors from an entire epoch the weights would be 0.4, 0.4, 0.1. If we plot these weights on the graph we get (again the line is *roughly* in the correct position).



And we can see that the lineraly separability lies is in a valid position and this can be confirmed by checking that the neural network is producing the correct outputs. You might like to do this for weights of 0.4, 0.4 and 0.1.

As I said above the perceptron spreadsheet implements the discussion from above. You might like to play around with the various values. In particular, see if you can get this spreadsheet to learn the XOR function.

References

- Aleksander, I., Morton, H. 1995. An Introduction to Neural Computing (2nd ed). Chapman and Hall
- Anderson, J.A., Rosenfeld, E. (eds). 1988. Neurocomputing: Foundations of Research. Cambridge, MA: MIT Press
- Block, H.D. 1962. The Perceptron: A Model for Brain Functioning, I. Reviews of Modern Physics, Vol 34, pp 123-135. Reprinted in Anderson and Rosenfeld,1988, pp 138-150
- Bryson, A.E., Ho, Y-C. 1969. Applied Optimal Control. New York: Blaisdell
- Callan, R. (1999) The Essence of Neural Networks, Prentice Hall, ISBN 0-13-908732-X
- Davalo, E., Naim, P. (1991). Neural Networks, The Macmillan Press Ltd, ISBN 0-333-54996-1
- Fausett, L. 1994. Fundamentals of Neural Networks : Architectures, Algorithms and Applications. Prentice-Hall, ISBN 0-13-103805-2
- Hebb, D.O. 1949. The Organization of Behavior. New York: John Wiley & Sons. Introduction and Chapter 4 reprinted in Anderson & Rosenfeld, 1988, pp 45-56
- Le Cun, Y. 1986. Learning Processes in an Asymmetric Theshold Network. Disordered Systems and Biological Organization (Bienenstock, E., Fogelman-Smith, F., Weisbuch, G. (eds)), NATO ASI Series, F20, Berlin: Springer-Verlag
- McCulloch, W.S., Pitts, W. 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics, Vol 5, pp 115-133. Reprinted in Anderson & Rosenfeld, 1988, pp 18-28.
- Minsky, M.L., Papert, S.A. 1988. Perceptrons, Expanded Edition. Cambridge, MA: MIT Press. Original Edition, 1969.
- Parker, D. 1985. Learning Logic. Technical Report TR-87, Cambridge, MA: Center for Computational Research in Economics and Management Science, MIT
- Rosenblatt, F. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review, Vol 65, pp 386-408. Reprinted in Anderson and Rosenfeld, 1988, pp 92-114
- Rosenblatt, F. 195. Two Theorems of Statistical Separability in the Perceptron. Mechanzization of Thought Processes of a Symposium held at the National Physical Laboratory, November, 1958. London: HM Stationery Office, pp 421-456
- Rosenblatt, F. 1962. Principles of Neurodynamics. New York: Spartan
- Russell, S., Norvig, P. 1995. Artificial Intelligence A Modern Approach. Prentice-Hall. ISBN 0-13-103805-2
- Werbos, P. 1974. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D

Thesis, Cambridge, MA: Harvard U. Committee on Applied Mathematics.

Last Updated : 21 Sep 2001