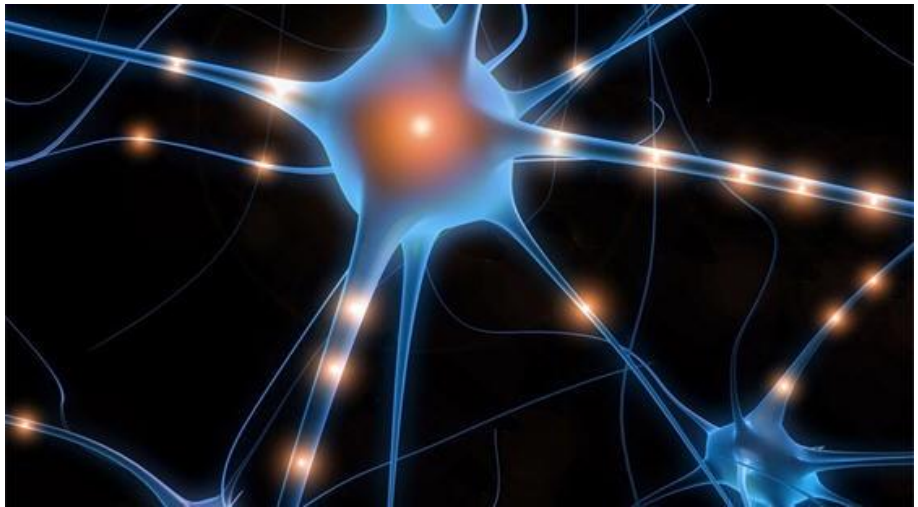
Introduction to Artificial Intelligence (G51IAI)

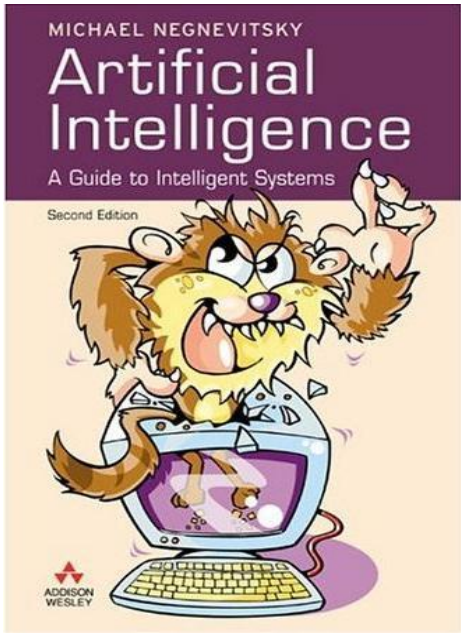
Dr Rong Qu

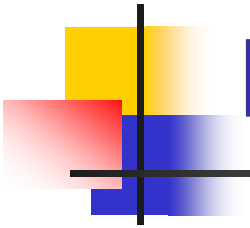
Neural Networks



Neural Networks

- Chapter 20 – **Artificial Intelligence : A Modern Approach** (**AIMA)**
  Russell and Norvig
- Chapter 6 – **Artificial intelligence: a guide to intelligent systems** (**AIIS**)
  Negnevitsky

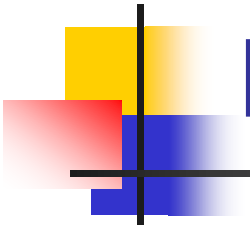




Neural Networks

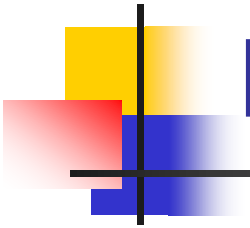
- More precisely: Artificial Neural Networks
  - Simulating, on a computer, what we understand about neural networks in the brain

- Learning Objectives
  - First of all, a brief history of work in ANN
  - Important concepts/terms in ANN
  - Then look at how simple algebra problems can be solved in ANN



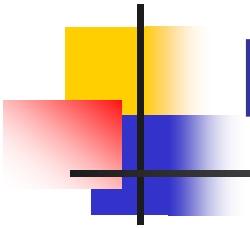
Neural Networks

- McCulloch & Pitts (1943) are generally recognised as the designers of the first neural network
  - Many simple units combine to give an increased computational power
  - The idea of a threshold
  - Many of their ideas still used today



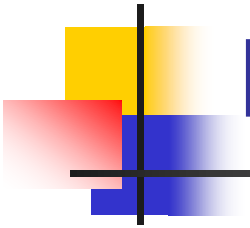
Neural Networks

- Hebb (1949) developed the first learning rule
  - McCulloch & Pitts network has fixed weights
  - If two neurons were active at the same time the strength between them should be increased



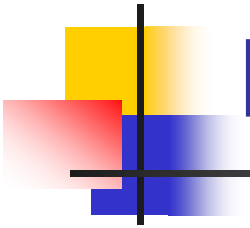
Neural Networks

- During the 50's and 60's
  - Many researchers worked on the perceptron, leading to great excitement
  - This model can be proved to converge to the correct weights
  - More powerful learning algorithm than Hebb



Neural Networks

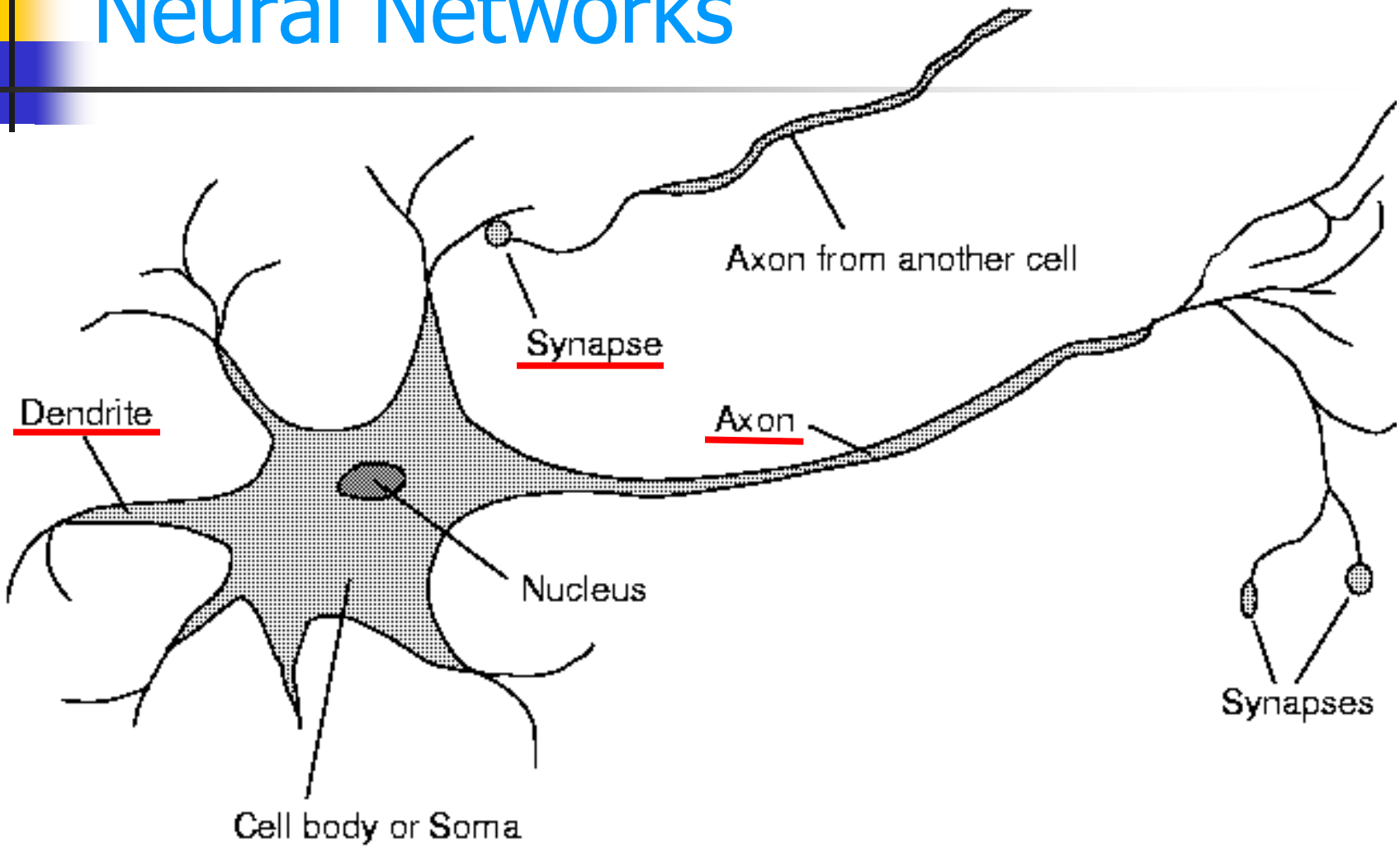
- 1969 saw the death of neural network research
  - Minsky & Papert
  - Perceptron can't learn certain type of important functions
  - Research of ANN went to decline for about 15 years



Neural Networks

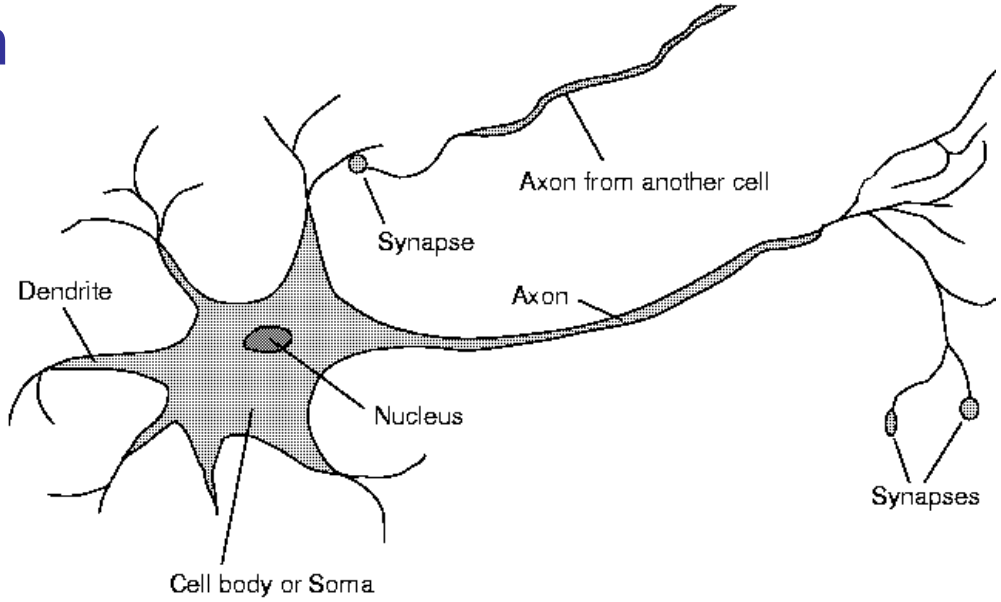
- Only in the mid 80's was interest revived
  - Parker (1985) and LeCun (1986) independently discovered multi-layer networks to solve problem of ***non-linear separable***
  - In fact Werbos discovered an algorithm in 1974, Bryson & Ho developed another algorithm in 1969

Neural Networks



Neural Networks

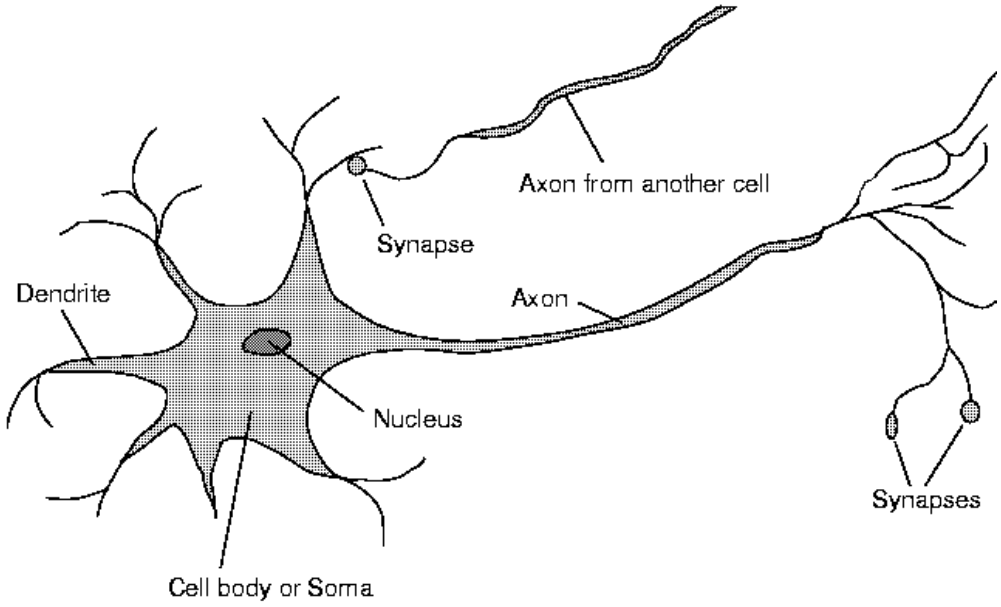
- We are born with about 100 billion neurons
- A neuron may connect to as many as 100,000 other neurons
- Many neurons die as we progress through life
- We continue to learn

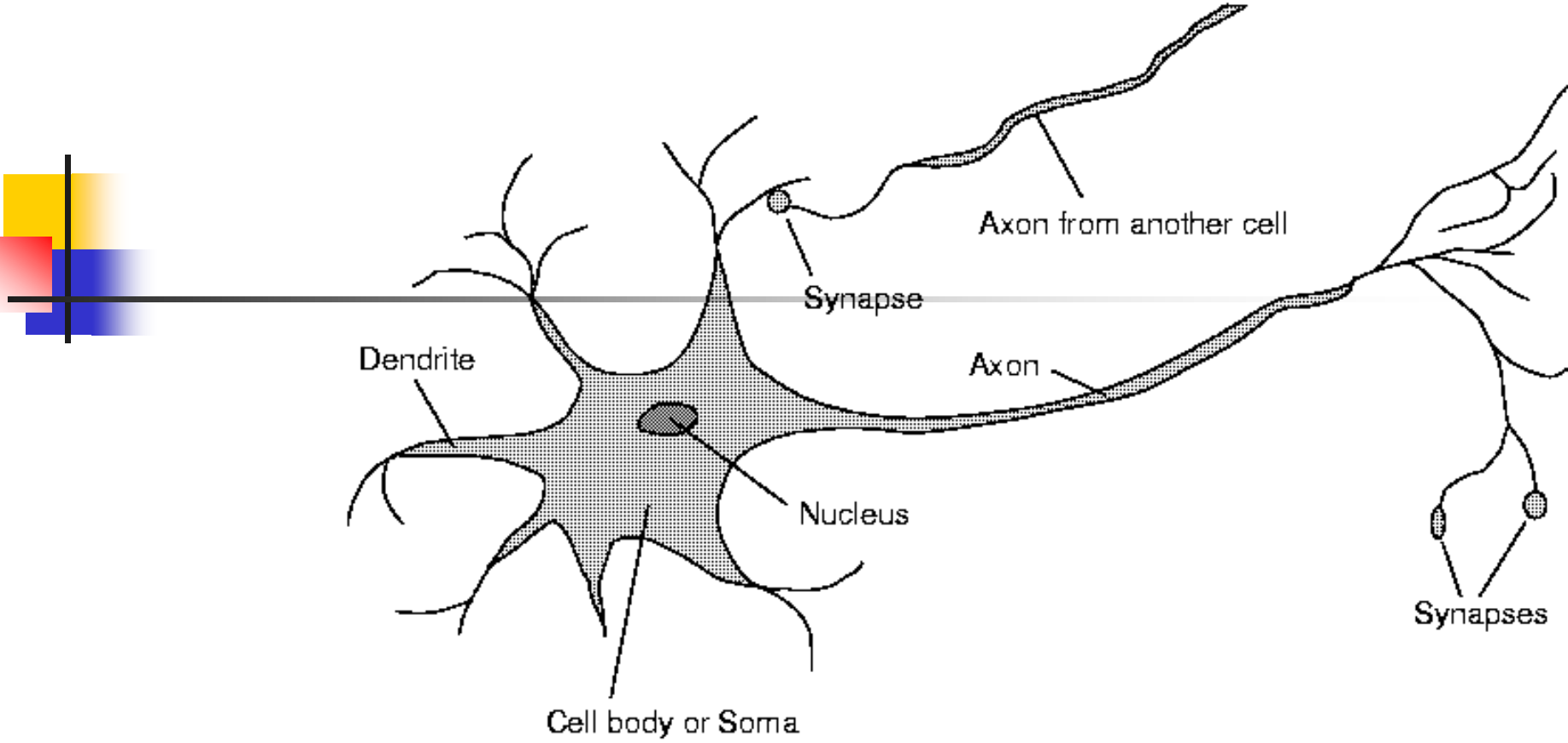


Neural Networks

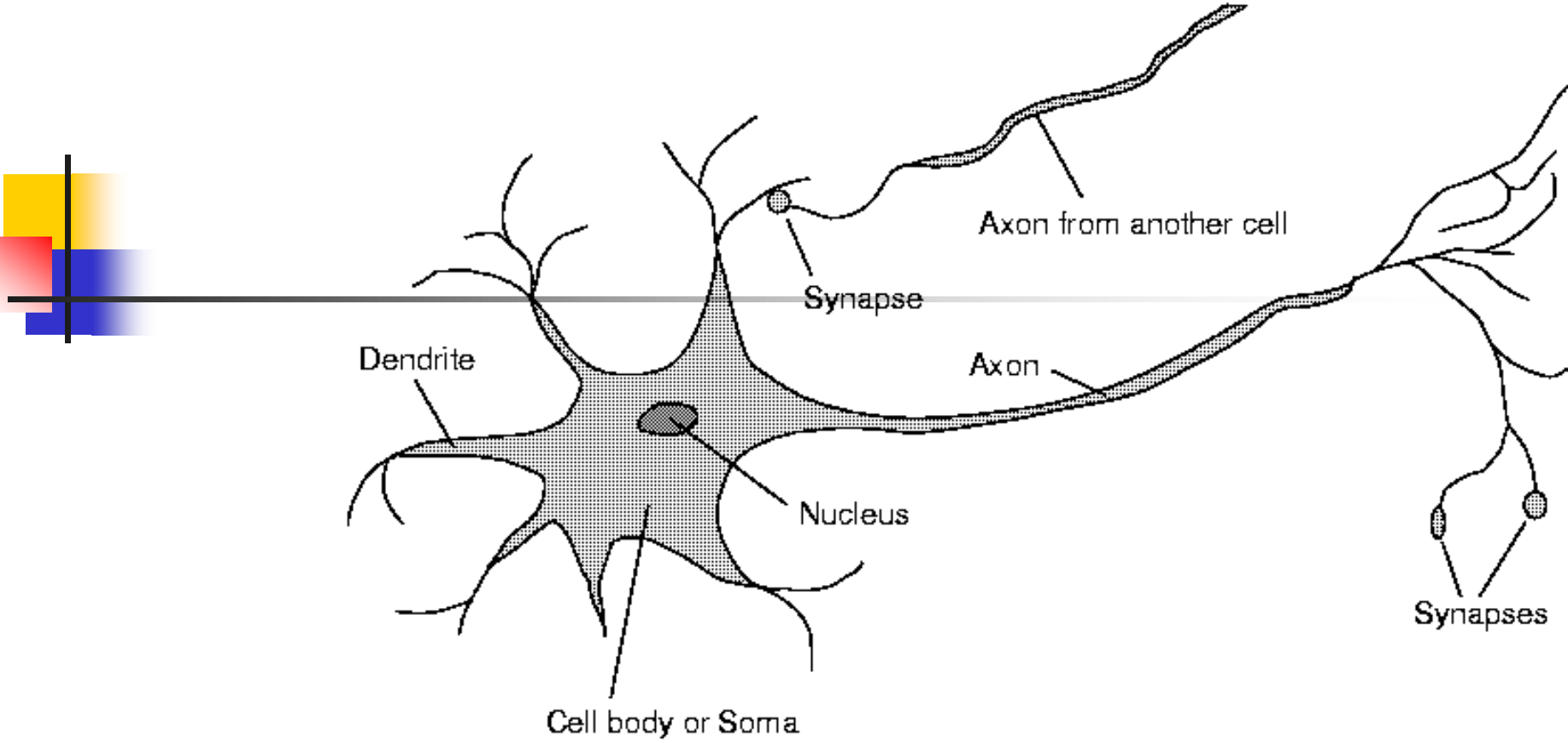
- Signals "move" via electrochemical reactions
- The synapses release a chemical transmitter – the sum of which can cause a threshold to be reached – causing the neuron to "fire"

- Synapses can be inhibitory or excitatory

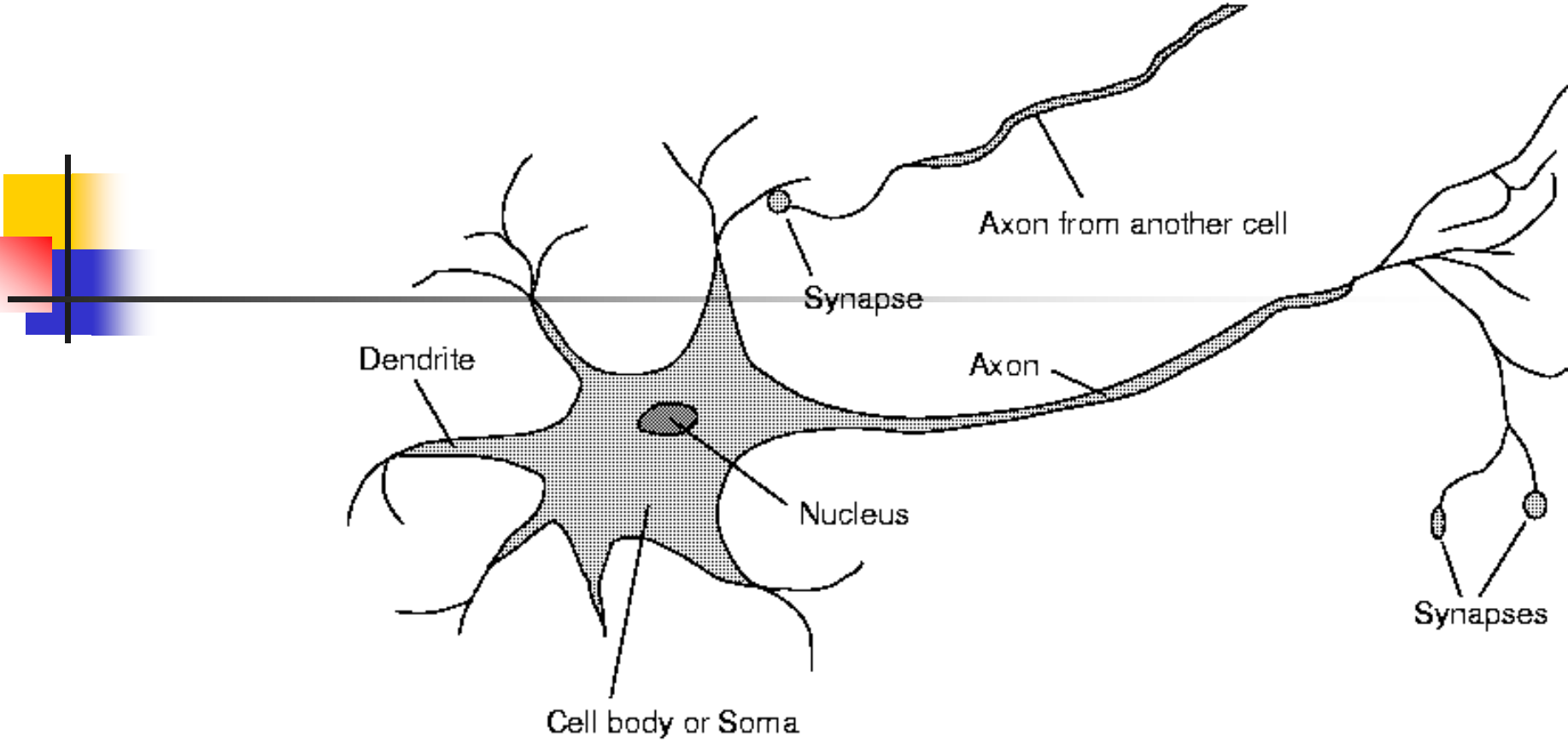




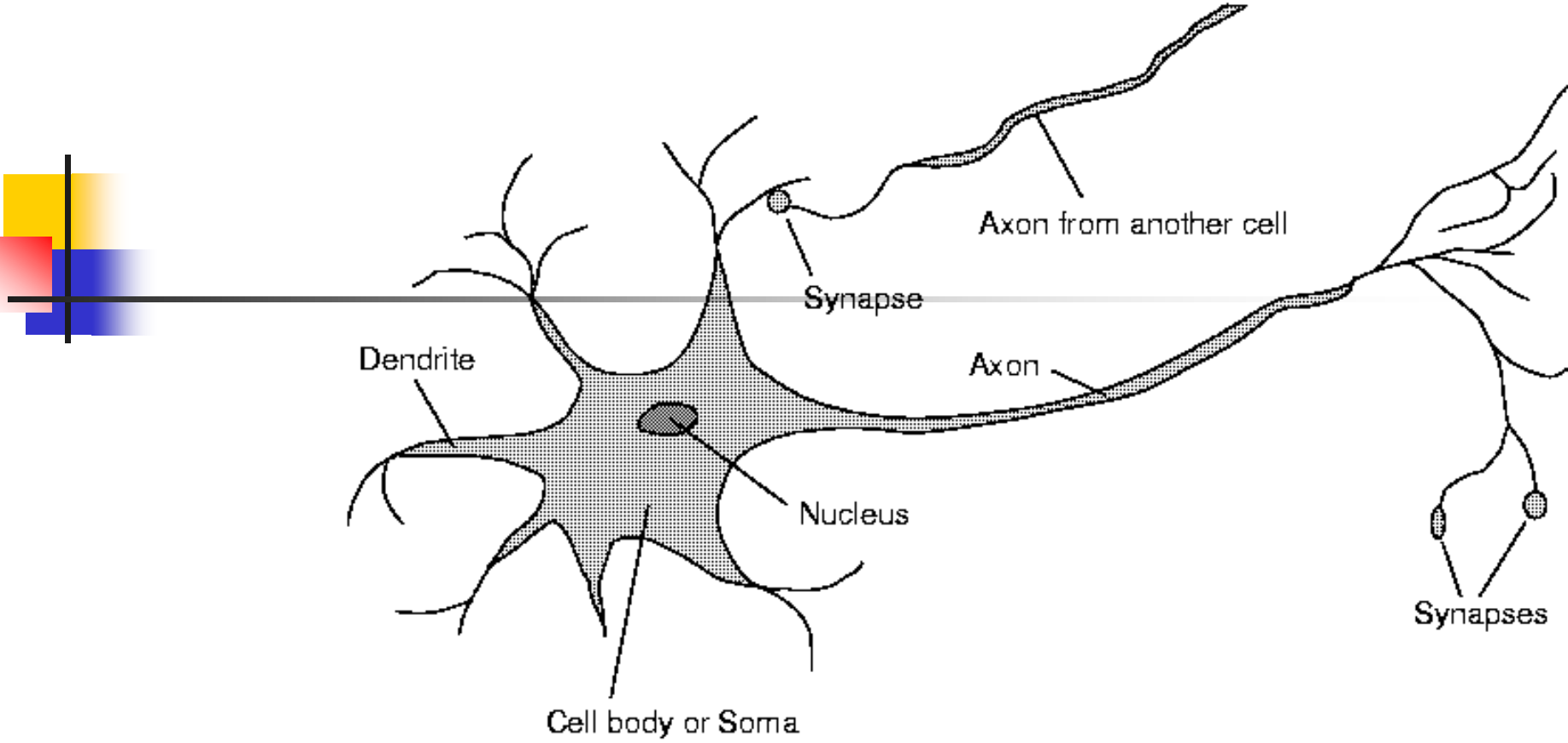
- From a computational point of view, the fundamental processing unit of a brain is a neuron



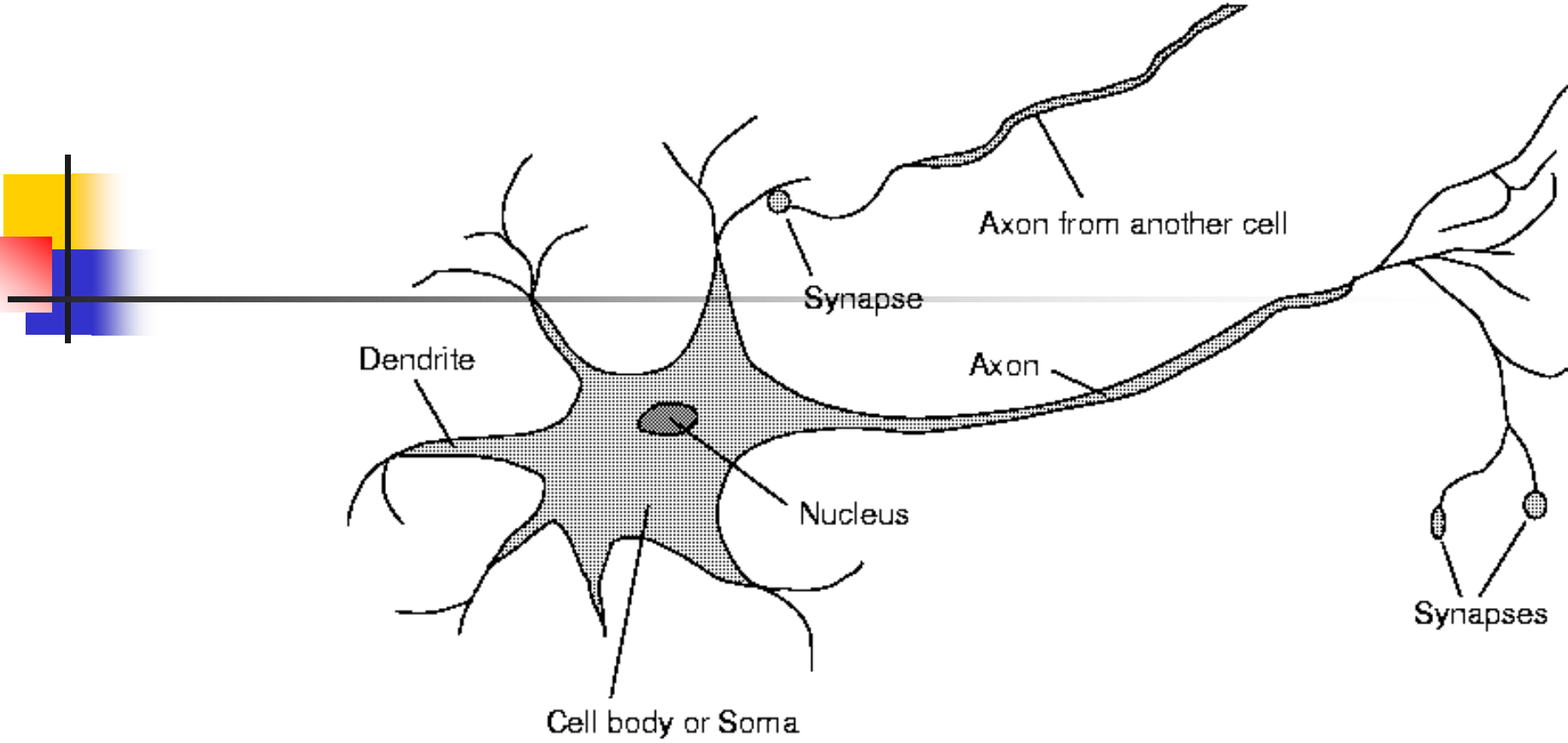
- A neuron consists of a cell body (soma with a nucleus)
- Each neuron has a number of dendrites which receive signals from other neurons



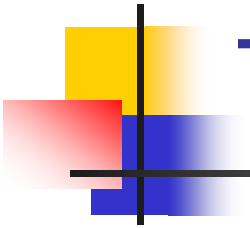
- Each neuron also has an axon goes out and splits into a number of strands to make a connection to other neurons
- The point at which neurons join other neurons is called a synapse



- Signals move between neurons 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)
- Long-term firing patterns formed – basic learning



The First Neural Networks

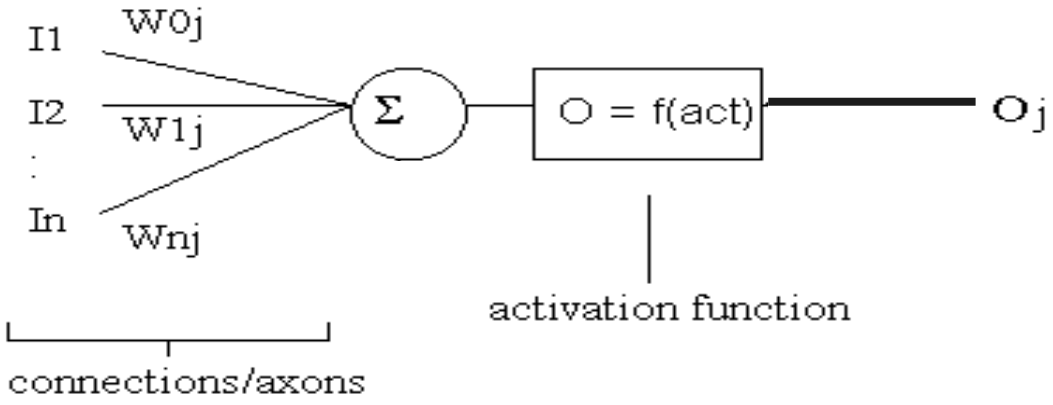
McCulloch and Pitts produced the first neural network in 1943

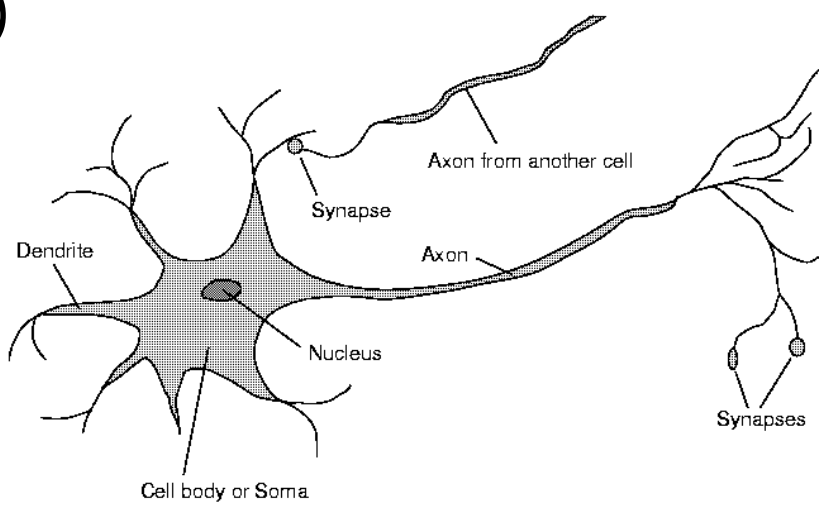
Many of the principles can still be seen in neural networks of today

The First Neural Networks

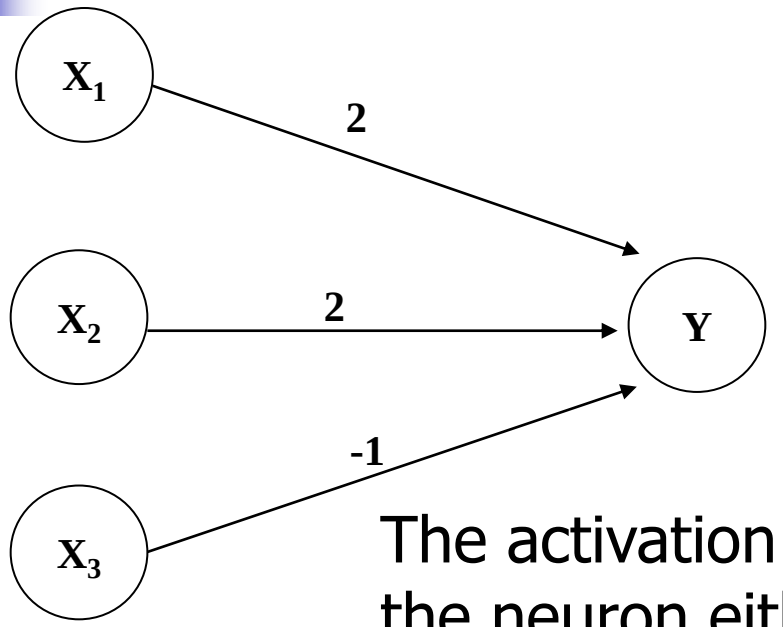
It consisted of:

- A set of inputs - (dendrites)
- A set of resistances/weights – (synapses)
- A processing element - (neuron)
- A single output - (axon)



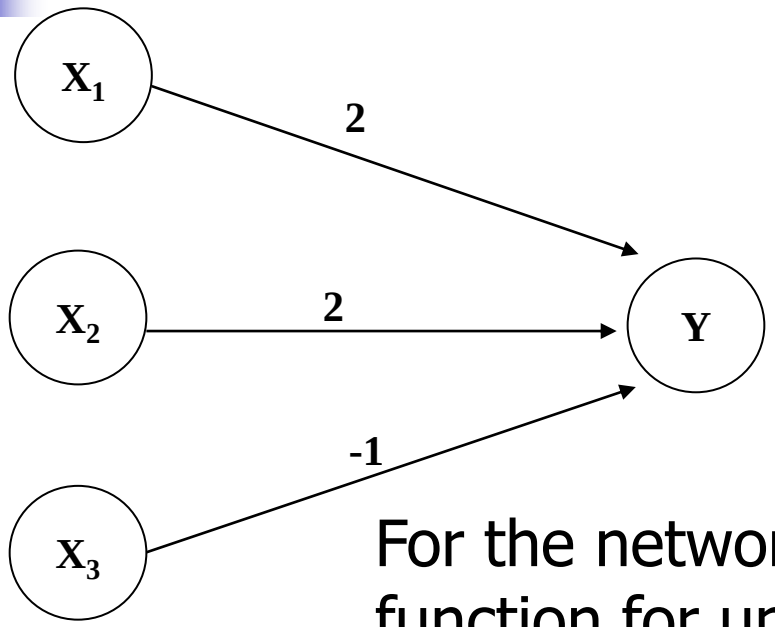


The First Neural Networks



The activation of a neuron is binary. That is, the neuron either fires (activation of one) or does not fire (activation of zero).

The First Neural Networks



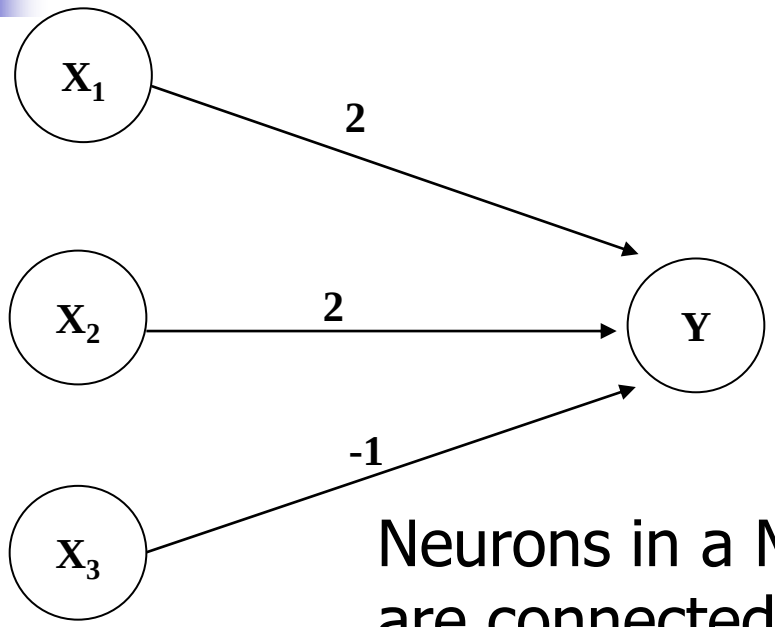
For the network shown here the activation function for unit *Y* is

f(y_in) = 1, if *y_in* >= θ; else 0

where *y_in* is the total input signal received;

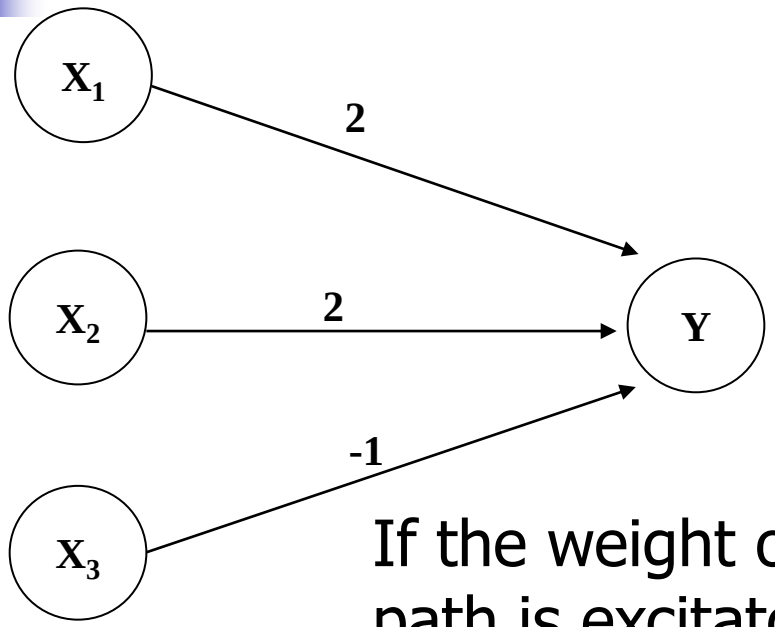
θ is the threshold for *Y*

The First Neural Networks



Neurons in a McCulloch-Pitts network are connected by directed, weighted paths

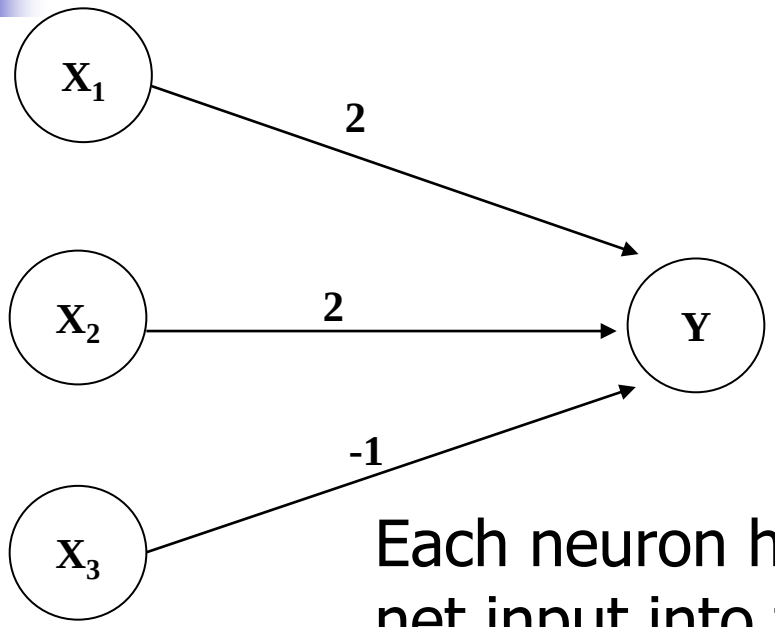
The First Neural Networks



If the weight on a path is positive the path is excitatory, otherwise it is inhibitory

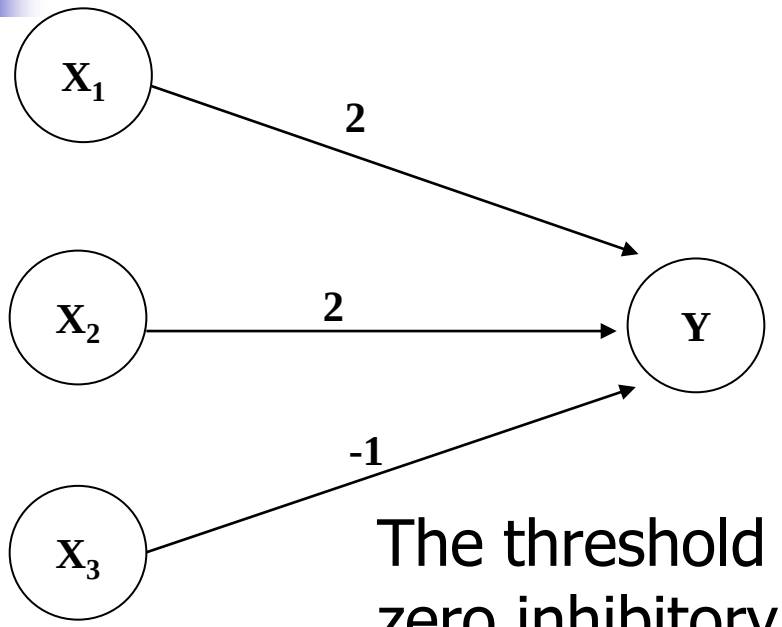
x_1 and x_2 encourage the neuron to fire
x_3 presents the neuron from firing

The First Neural Networks



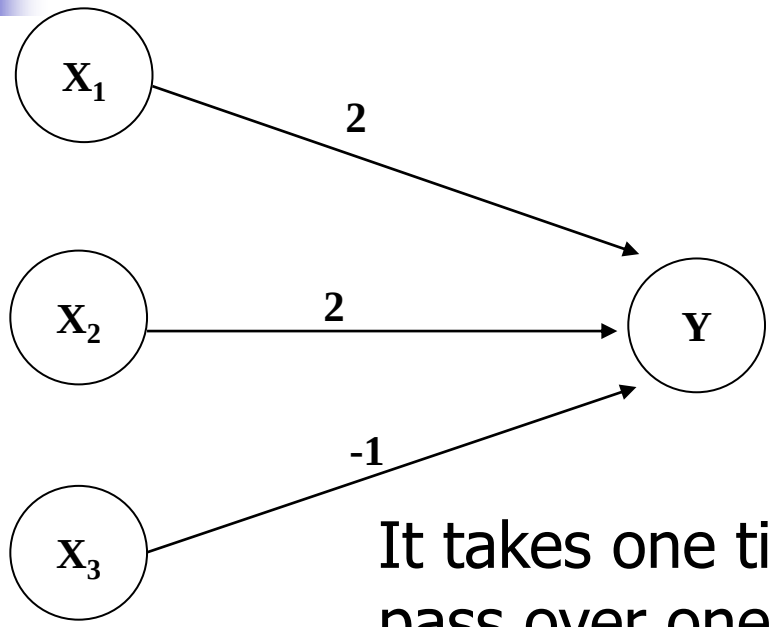
Each neuron has a fixed threshold. If the net input into the neuron is greater than or equal to the threshold, the neuron fires

The First Neural Networks



The threshold is set such that any non-zero inhibitory input will prevent the neuron from firing

The First Neural Networks

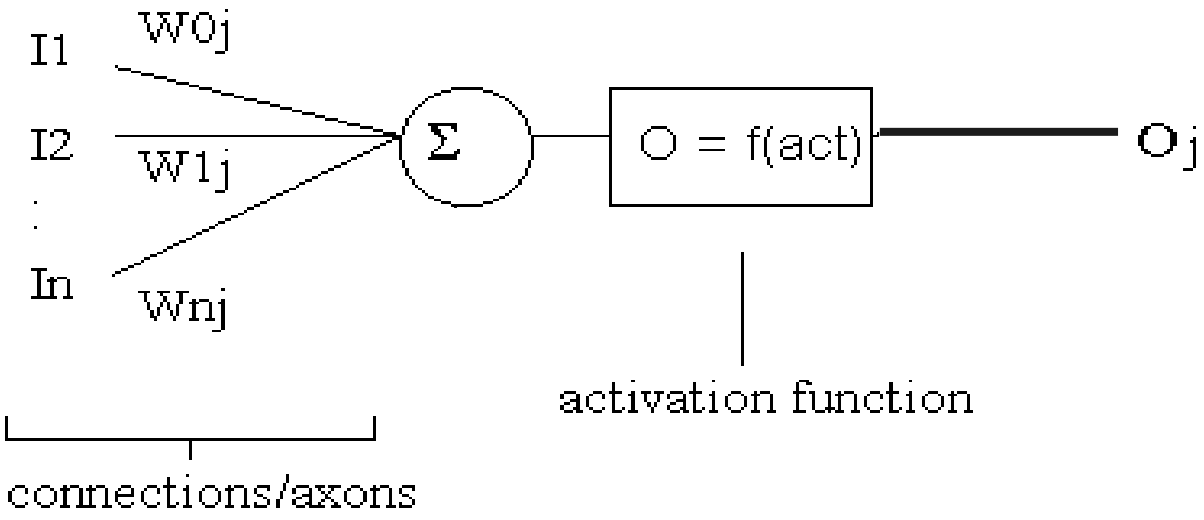


It takes one time step for a signal to pass over one connection.

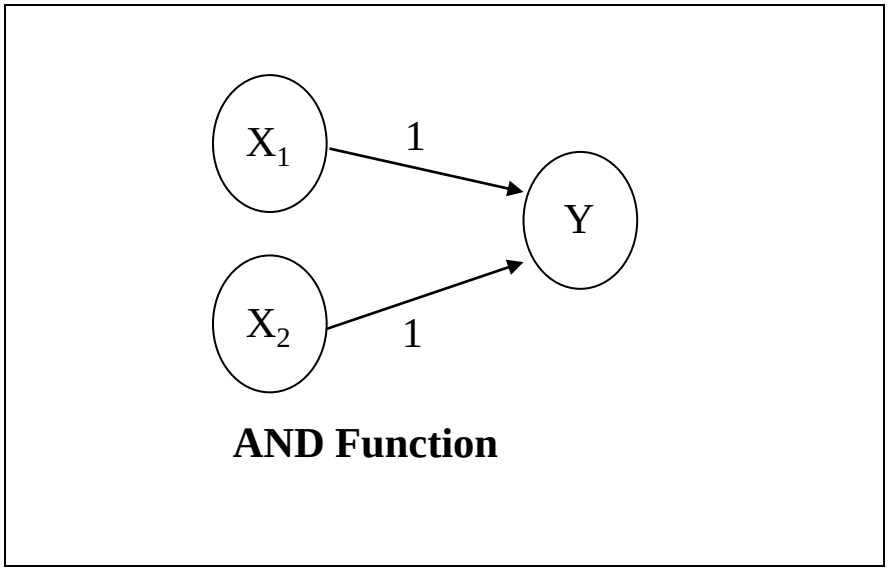
The First Neural Networks

Using McCulloch-Pitts model we can model logic functions

Let's look at 4 logic functions



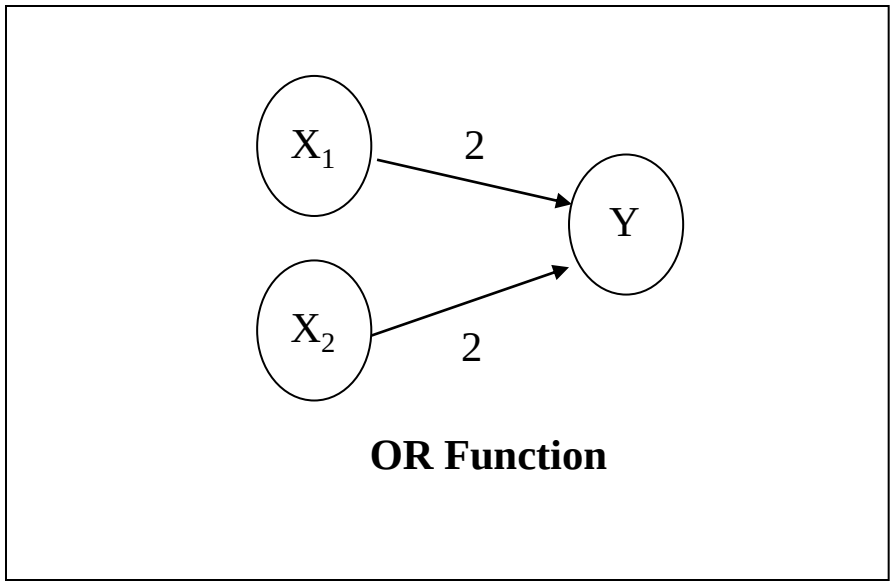
The First Neural Networks



AND		
X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0

Threshold(Y) = 2

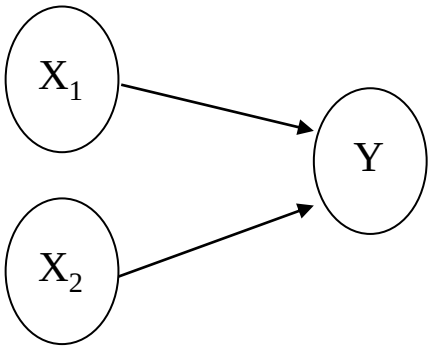
The First Neural Networks



OR		
X1	X2	Y
1	1	1
1	0	1
0	1	1
0	0	0

Threshold(*Y*) = 2

The First Neural Networks

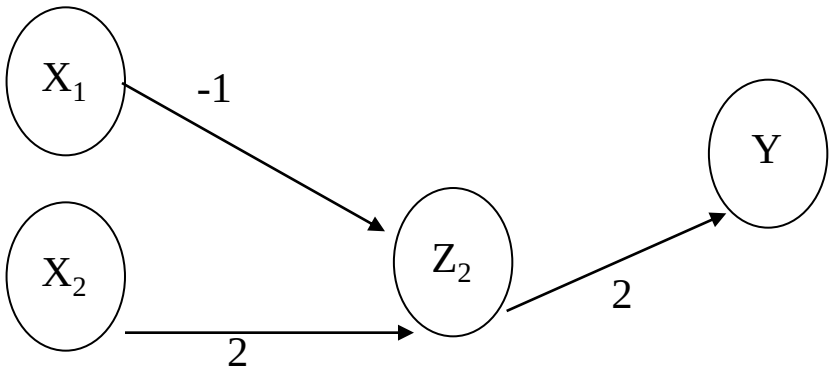


AND NOT Function

AND NOT		
X1	**X2**	**Y**
1	1	0
1	0	1
0	1	0
0	0	0

Threshold(*Y*) = 2

The First Neural Networks

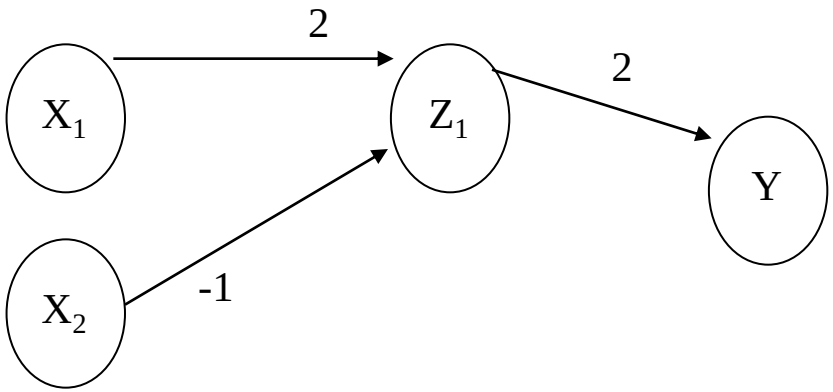


XOR		
X1	X2	Y
1	1	0
1	0	1
0	1	1
0	0	0

AND NOT		
X1	X2	Y
1	1	0
1	0	1
0	1	0
0	0	0

X_1 XOR X_2 = (X_1 AND NOT X_2) OR (X_2 AND NOT X_1)

The First Neural Networks

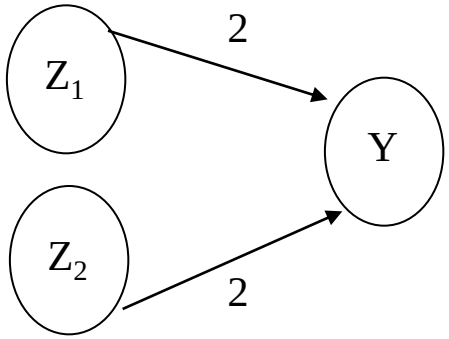


XOR		
X1	X2	Y
1	1	0
1	0	1
0	1	1
0	0	0

AND NOT		
X1	X2	Y
1	1	0
1	0	1
0	1	0
0	0	0

X_1 XOR X_2 = (X_1 AND NOT X_2) OR (X_2 AND NOT X_1)

The First Neural Networks



XOR		
X1	X2	Y
1	1	0
1	0	1
0	1	1
0	0	0

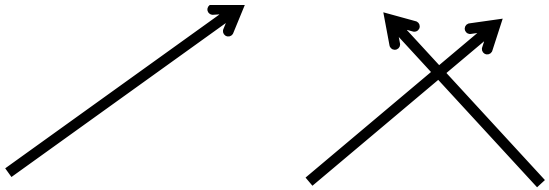
OR		
X1	X2	Y
1	1	1
1	0	1
0	1	1
0	0	0

X_1 XOR X_2 = (X_1 AND NOT X_2) OR (X_2 AND NOT X_1)

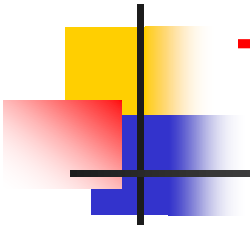
The First Neural Networks

XOR	
X1	**X2**
1	1
1	0
0	1
0	0

Y
0
1
1
0



X_1 XOR X_2 = (X_1 AND NOT X_2) OR (X_2 AND NOT X_1)



The First Neural Networks

Another example: how to model

To model this we will assume that time is discrete

If we touch something cold we initially perceive heat

If cold is applied for one time step then heat will be perceived

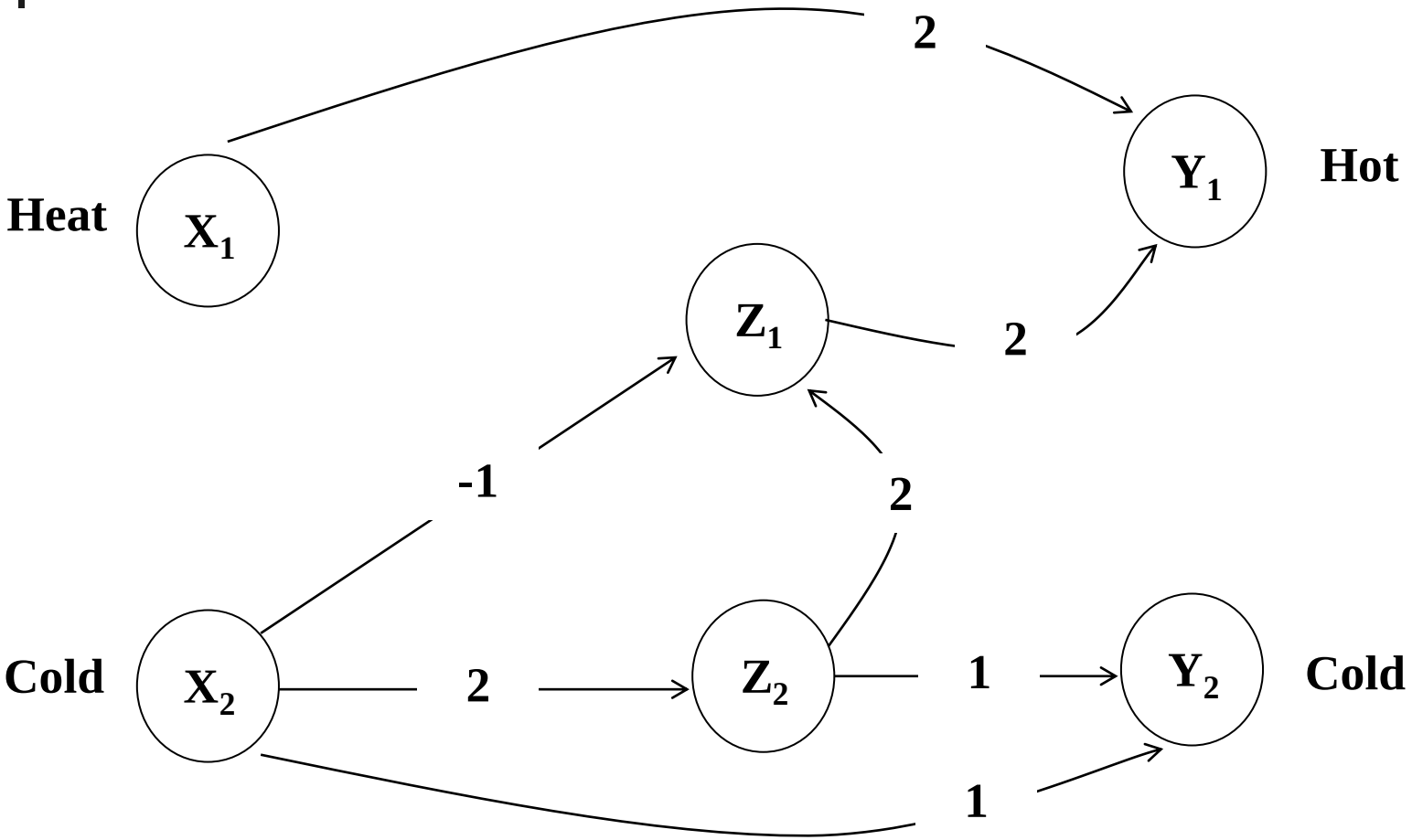
If we keep touching something cold we will perceive cold

If a cold stimulus is applied for two time steps then cold will be perceived

If we touch something hot we will perceive heat

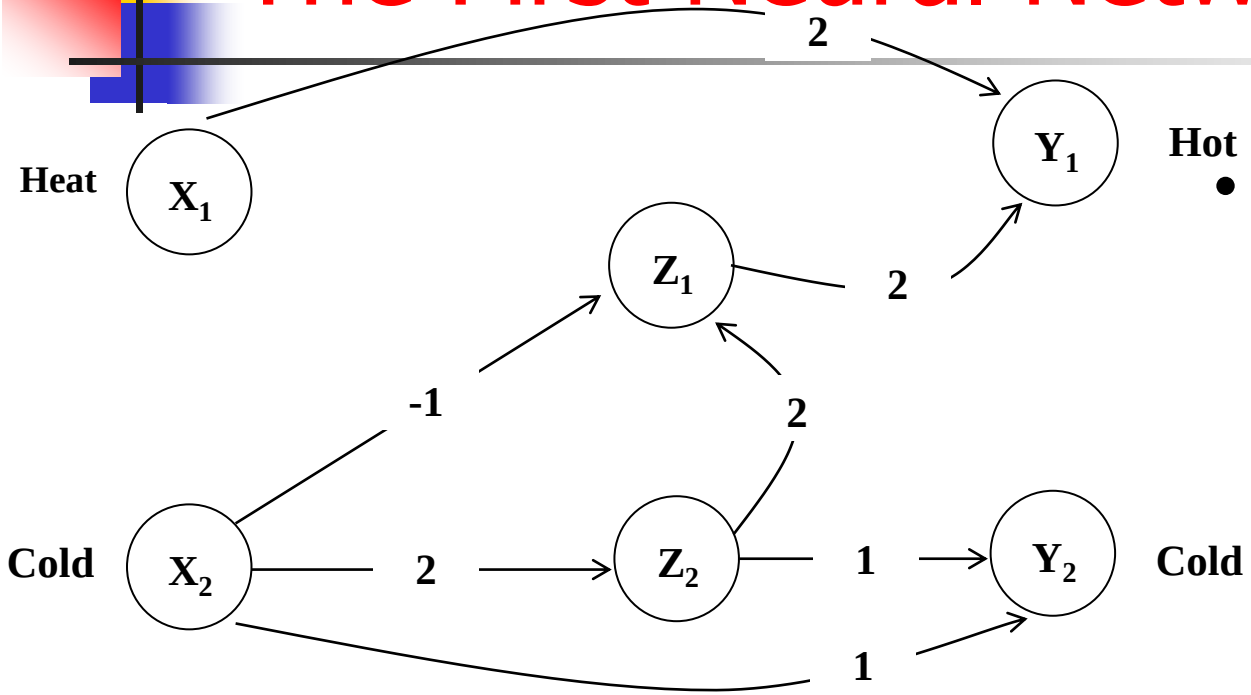
If heat is applied then we should perceive heat

The First Neural Networks



Threshold(Y) = 2

The First Neural Networks



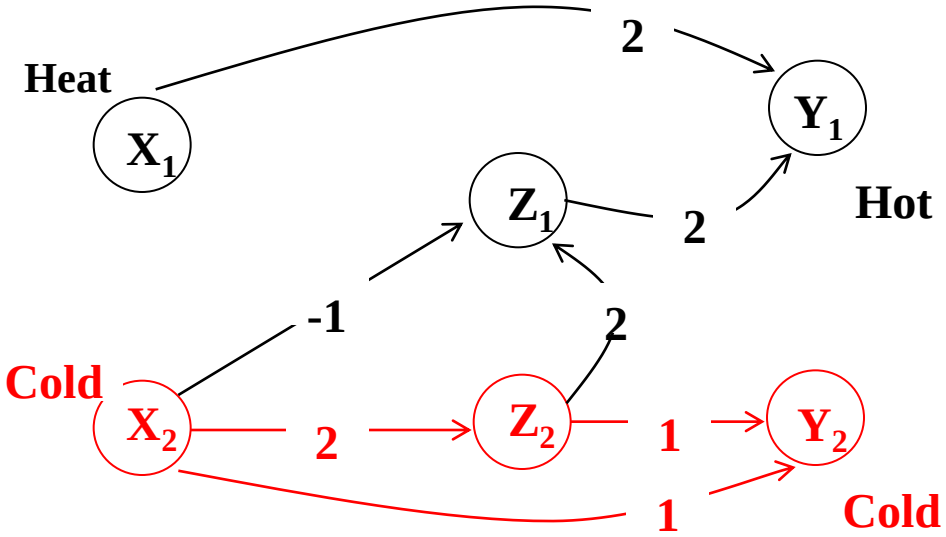
- It takes time for the stimulus (applied at X_1 and X_2) to make its way to Y_1 and Y_2 where we perceive either heat or cold

- 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

The First Neural Networks

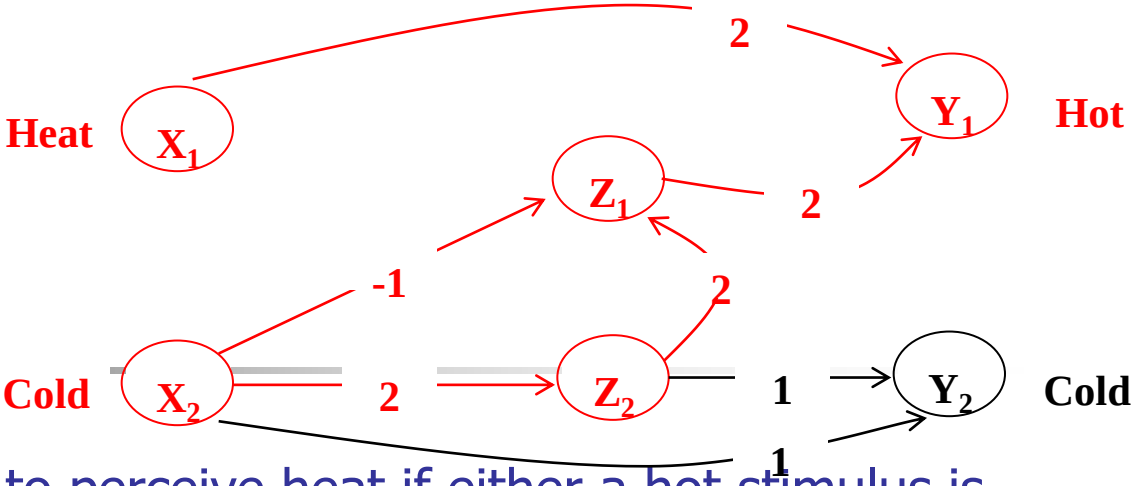
We want the system to perceive cold if a cold stimulus is applied for two time steps

$$Y_2(t) = X_2(t-2) \text{ AND } X_2(t-1)$$



For all neurons, threshold = 2

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

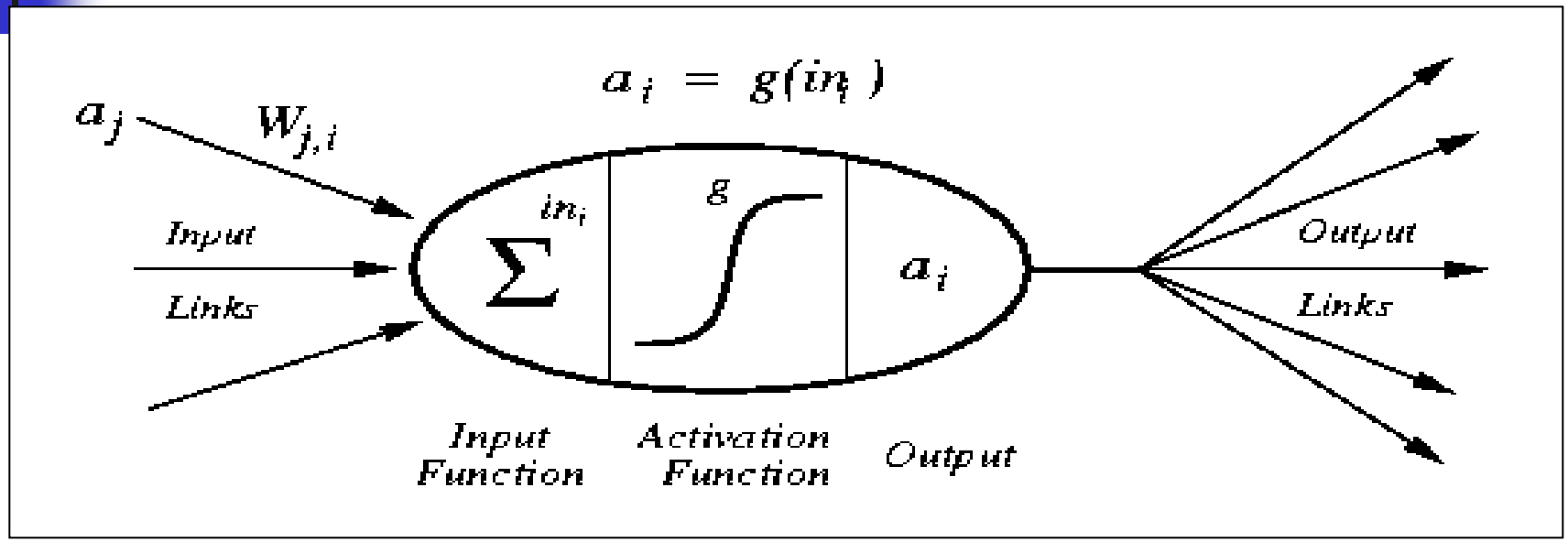


We want the system to perceive heat if either a hot stimulus is applied or a cold stimulus is applied (for one time step) and then removed

$$Y_1(t) = [\, X_1(t-1) \,] \text{ OR } [\, X_2(t-3) \text{ AND NOT } X_2(t-2) \,]$$

X2(t – 3)	X2(t – 2)	X1(t – 1)
1	1	1
1	0	1
0	1	1
0	0	1
1	1	0
1	0	0
0	1	0
0	0	0

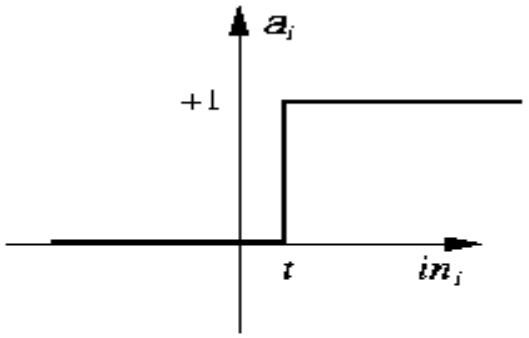
Modelling a Neuron



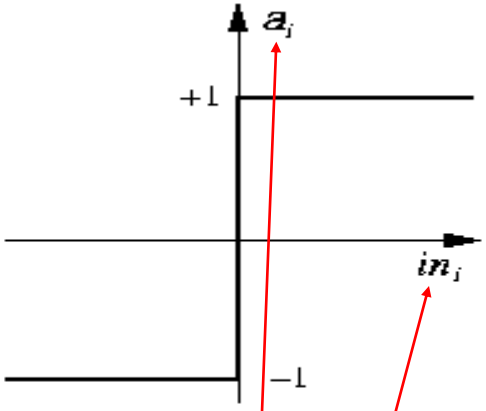
$$in_i = \sum_j W_{j,i} a_j$$

- a_j : Input value (output from 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
- g : Activation function

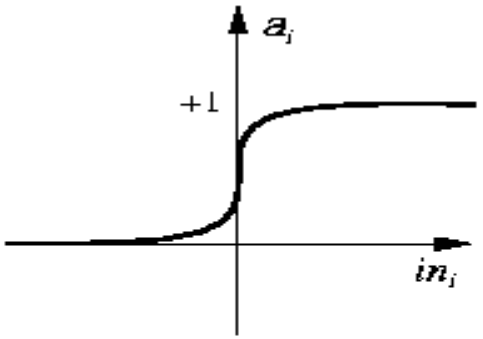
Activation Functions



(a) Step function



(b) Sign function



(c) Sigmoid function

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

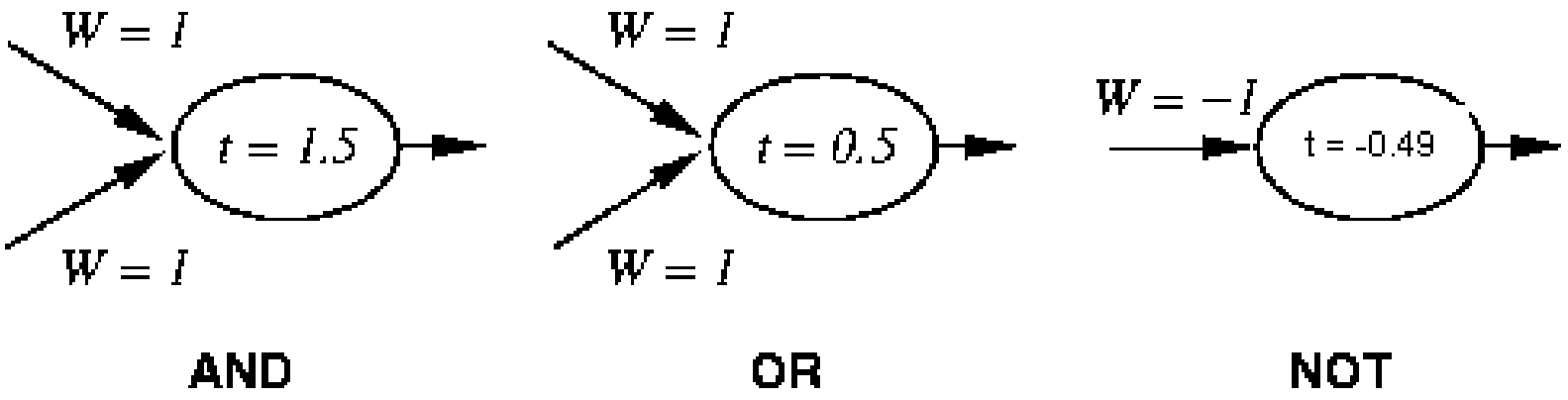
- a_j : Input value (output from unit j)
- in_i : Weighted sum of inputs to unit i
- a_i : Activation value of unit i
- g : Activation function

Simple Networks

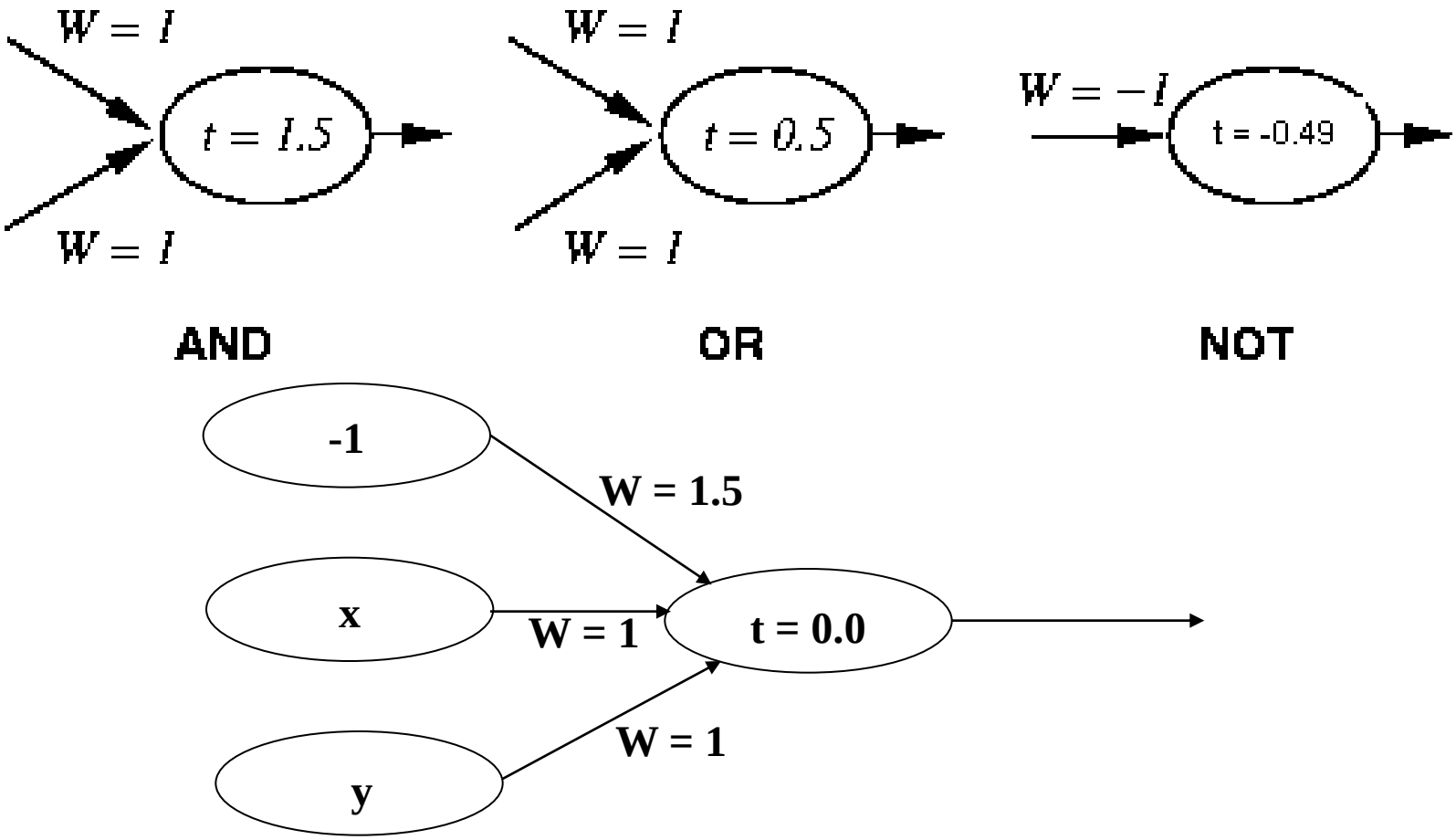
	AND			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	0	1

	OR			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	1	1	1

	NOT	
Input 1	0	1
Input 2		
Output	1	0



Simple Networks

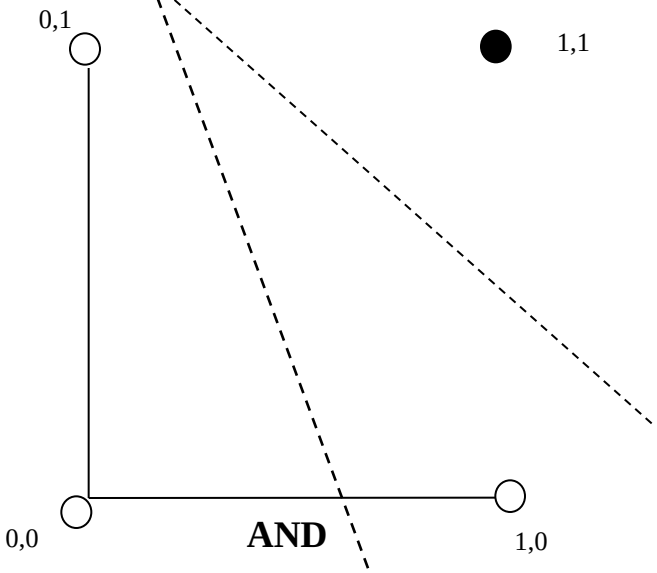


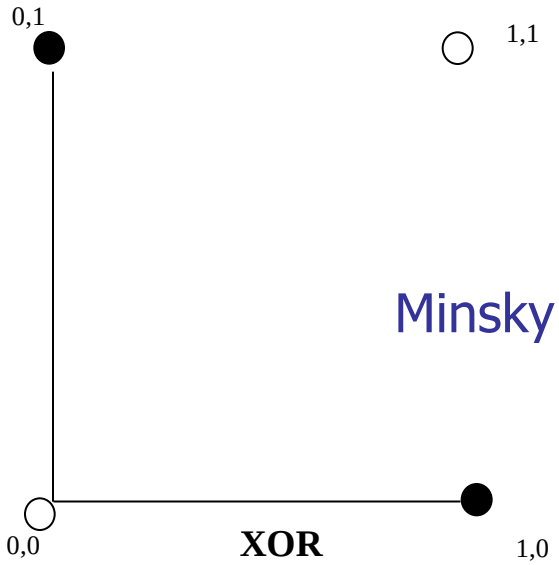
What can Single Layer NN represent?

	AND			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	0	1

	XOR			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	1	1	0

	AND				XOR			
Input 1	0	0	1	1	0	0	1	1
Input 2	0	1	0	1	0	1	0	1
Output	0	0	0	1	0	1	1	0





Minsky & Papert

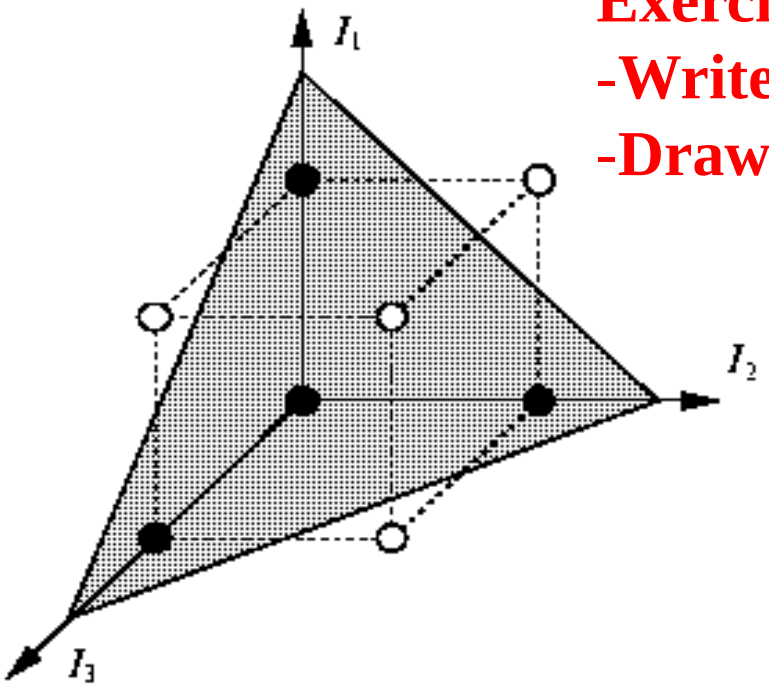
- Functions which can be separated in this way are called *Linearly Separable*
- Only linearly Separable functions can be represented by a single layer NN

What can Single Layer NN represent?

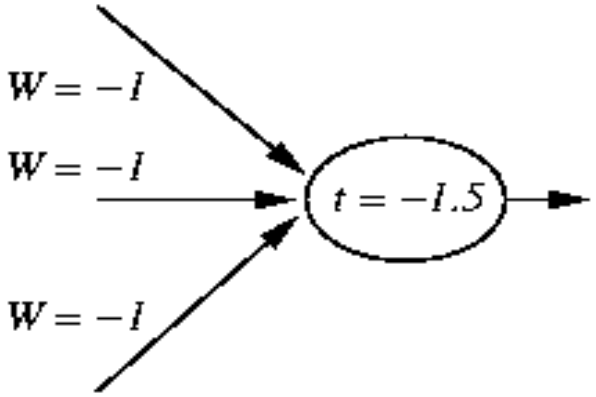
Exercise:

-Write the truth table

-Draw 2-D graphical representation



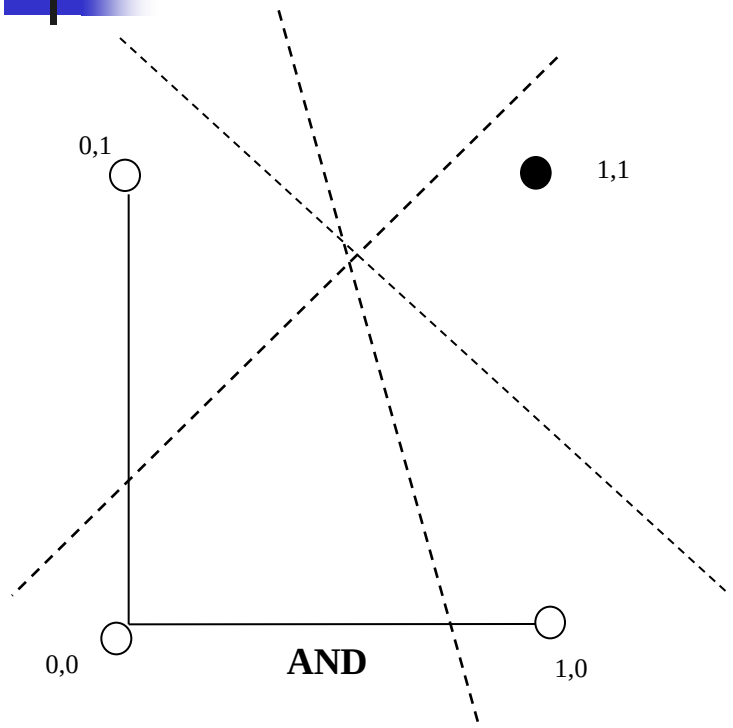
(a) Separating plane



(b) Weights and threshold

Linear Separability is also possible in more than 3 dimensions – but it is harder to visualise

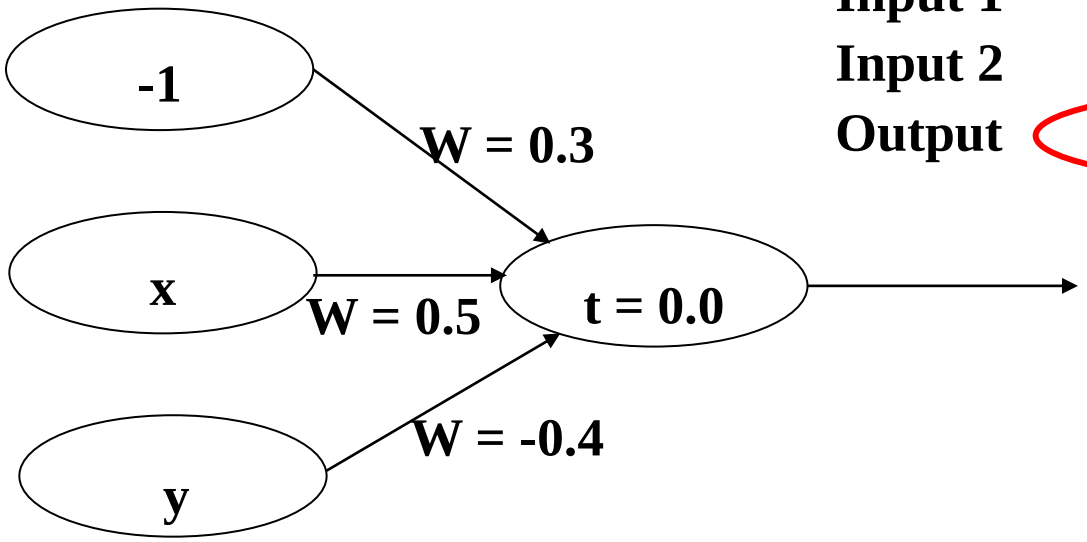
Training a NN



	AND			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	0	1

Training a NN

AND			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	0	1



I_1	I_2	I_3	Summation	Output
-1	0	0	(-1*0.3) + (0*0.5) + (0*-0.4) = -0.3	0
-1	0	1	(-1*0.3) + (0*0.5) + (1*-0.4) = -0.7	0
-1	1	0	(-1*0.3) + (1*0.5) + (0*-0.4) = 0.2	1
-1	1	1	(-1*0.3) + (1*0.5) + (1*-0.4) = -0.2	0

Learning

AND			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	0	1

0	0	1	0

While epoch produces an error

Present network with next inputs (pattern) from epoch

Err = T – O

If Err <> 0 then

$W_j = W_j + LR * I_j * Err$

End If

End While

Learning

AND			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	0	1

0	0	1	0

While epoch produces an error

Present network with next inputs (pattern) from epoch

Err = T – O

If Err <> 0 then

$W_j = W_j + LR * I_j * Err$

End If

End While

Epoch : 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 (patterns) being presented to the network (i.e. [0,0], [0,1], [1,0], [1,1])

Learning

AND			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	0	1

0	0	1	0

While epoch produces an error
    Present network with next inputs (pattern) from epoch
    Err = T – O
    If Err <> 0 then
        $W_j = W_j + LR * I_j * Err$
    End If
End While

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

Learning

AND			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	0	1

0	0	1	0

While epoch produces an error
    Present network with next inputs (pattern) from epoch
    Err = T – O
    If Err <> 0 then
        $W_j = W_j + LR * I_j * Err$
    End If
End While

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

Learning

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

0	0	1	0

While epoch produces an error

Present network with next inputs (pattern) from epoch

Err = T – O

If Err <> 0 then

$W_j = W_j + LR * I_j * Err$

End If

End While

What is the problem if the learning rate is set too high, or too low?

Output from Neuron, O : The output value from the neuron

Ij : Inputs being presented to the neuron

Wj : Weight from input neuron (I_j) 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

Learning

AND			
Input 1	0	0	1	1
Input 2	0	1	0	1
Output	0	0	0	1

0	0	1	0

While epoch produces an error

Present network with next inputs (pattern) from epoch

Err = T – O

If Err <> 0 then

$W_j = W_j + LR * I_j * Err$

End If

End While

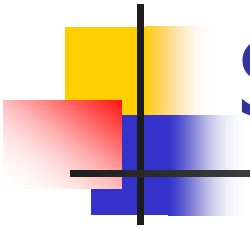
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Epoch	I0	I1	I2	Reqd Output	W0	W1	W2	Sum	Activation	Error	Converged?	Learning Rate
2	1	-1	0	0	0	0.3	0.5	-0.4	-0.3	0	0		0.1
3		-1	0	1	0	0.3	0.5	-0.4	-0.7	0	0		
4		-1	1	0	0	0.3	0.5	-0.4	0.2	1	-1		
5		-1	1	1	1	0.4	0.4	-0.4	-0.4	0	1	Not Converged	
6	2	-1	0	0	0	0.3	0.5	-0.3	-0.3	0	0		
7		-1	0	1	0	0.3	0.5	-0.3	-0.6	0	0		
8		-1	1	0	0	0.3	0.5	-0.3	0.2	1	-1		
9		-1	1	1	1	0.4	0.4	-0.3	-0.3	0	1	Not Converged	
10	3	-1	0	0	0	0.3	0.5	-0.2	-0.3	0	0		
11		-1	0	1	0	0.3	0.5	-0.2	-0.5	0	0		
12		-1	1	0	0	0.3	0.5	-0.2	0.2	1	-1		
13		-1	1	1	1	0.4	0.4	-0.2	-0.2	0	1	Not Converged	
14	4	-1	0	0	0	0.3	0.5	-0.1	-0.3	0	0		
15		-1	0	1	0	0.3	0.5	-0.1	-0.4	0	0		
16		-1	1	0	0	0.3	0.5	-0.1	0.2	1	-1		
17		-1	1	1	1	0.4	0.4	-0.1	-0.1	0	1	Not Converged	
18	5	-1	0	0	0	0.3	0.5	0	-0.3	0	0		
19		-1	0	1	0	0.3	0.5	0	-0.3	0	0		
20		-1	1	0	0	0.3	0.5	0	0.2	1	-1		
21		-1	1	1	1	0.4	0.4	0	0	0	1	Not Converged	
22	6	-1	0	0	0	0.3	0.5	0.1	-0.3	0	0		
23		-1	0	1	0	0.3	0.5	0.1	-0.2	0	0		
24		-1	1	0	0	0.3	0.5	0.1	0.2	1	-1		
25		-1	1	1	1	0.4	0.4	0.1	0.1	1	0	Not Converged	
26	7	-1	0	0	0	0.4	0.4	0.1	-0.4	0	0		
27		-1	0	1	0	0.4	0.4	0.1	-0.3	0	0		
28		-1	1	0	0	0.4	0.4	0.1	0	0	0		
29		-1	1	1	1	0.4	0.4	0.1	0.1	1	0	Converged	

And Finally….

"If the brain were so simple that we could understand it then we'd be so simple that we couldn't"

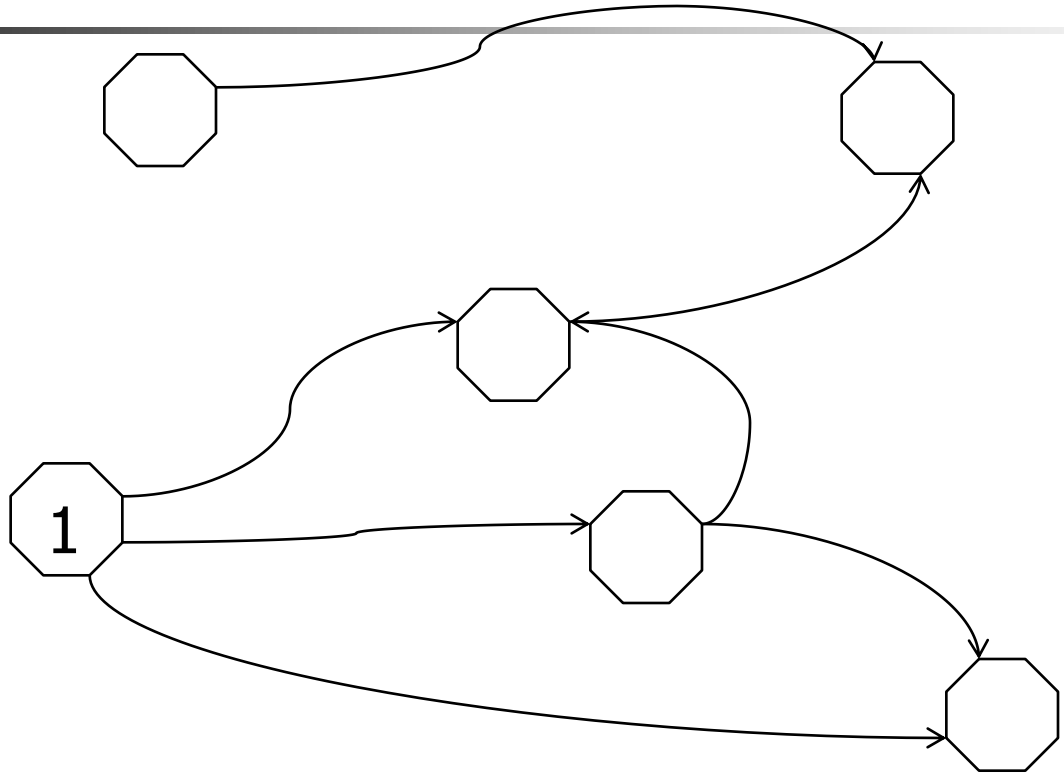
Lyall Watson

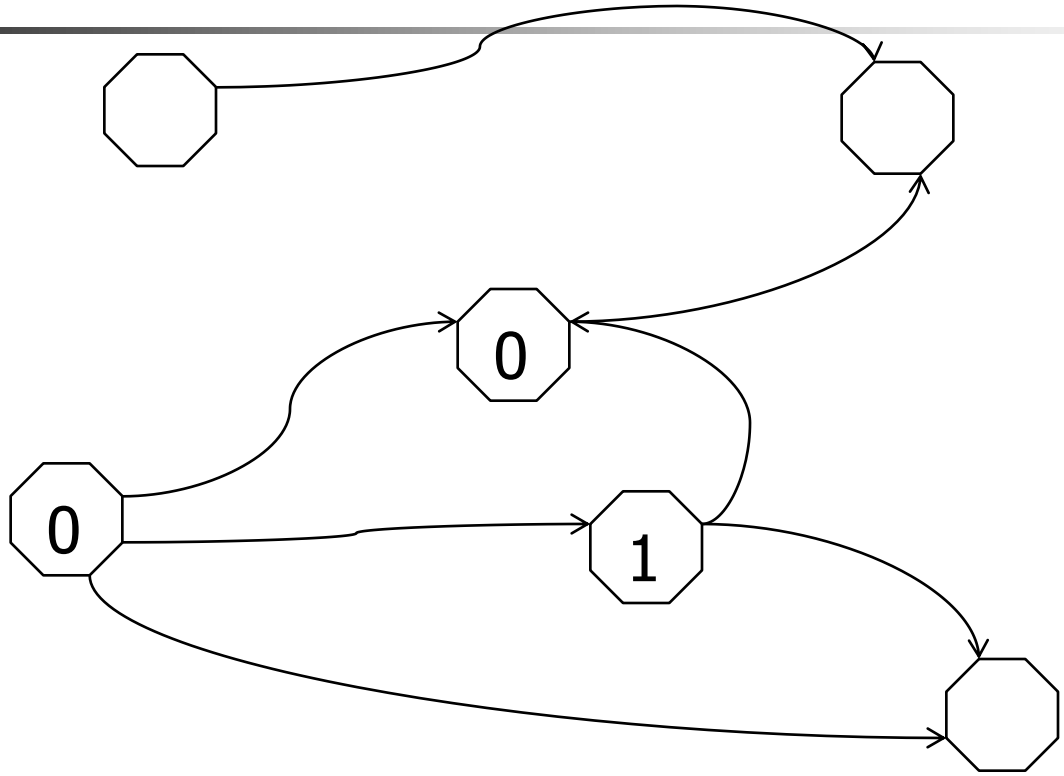
Can machines ever be intelligent?

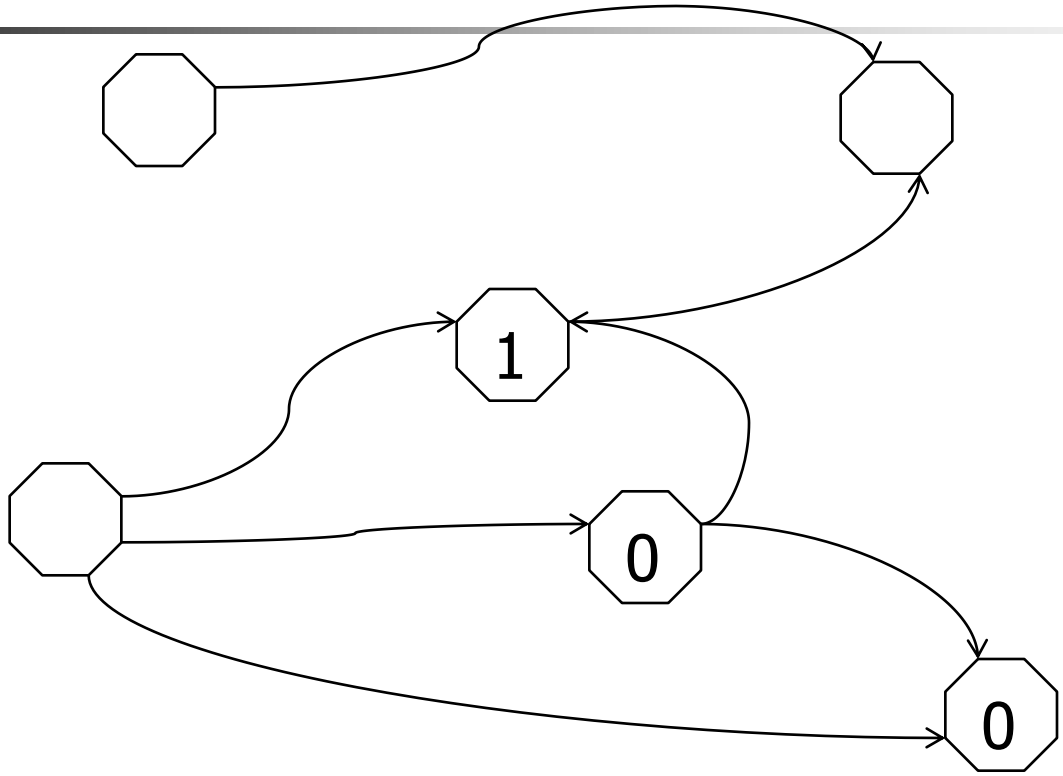


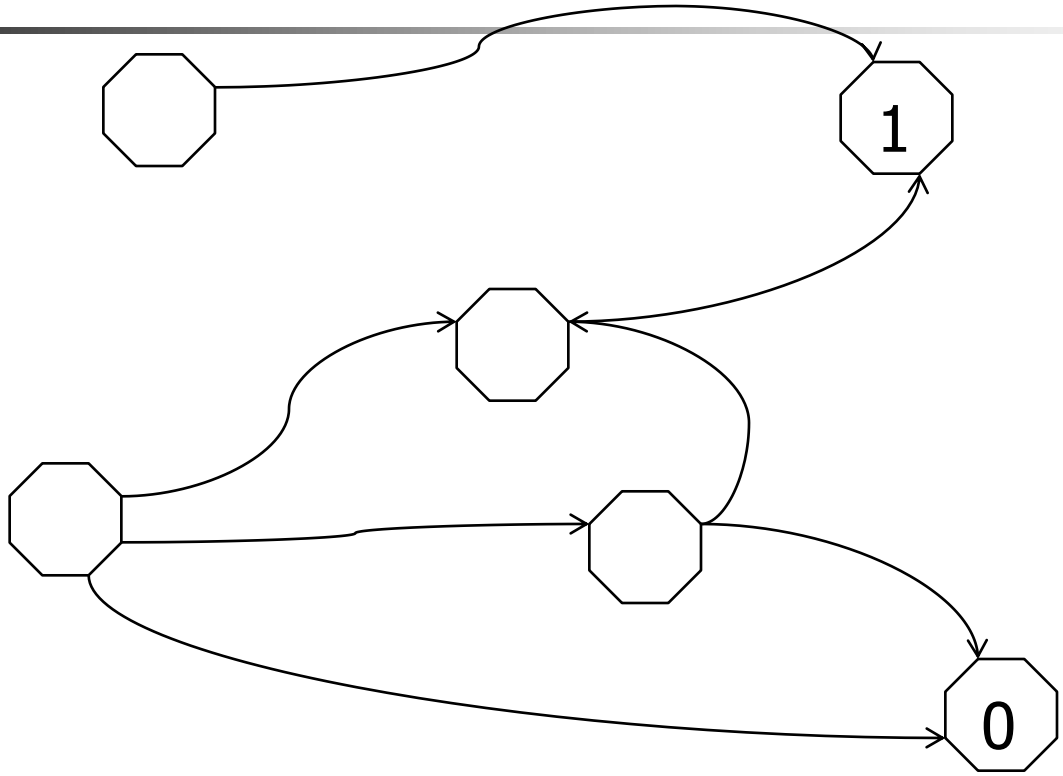
Summary

- First neural networks (McClloch-Pitts)
- Simple networks
- Perceptron (single layer NN)
  - Representation
  - Limitations (linearly separable)
  - Learning


1


0
0
1


1
0
0


1
0