Introduction to Neural Networks
Lecture 1 of 3

Kevin Sim

Slides courtesy of Prof. Emma Hart

Next 3 Lectures

1. Background – simple perceptrons
2. Multi-layer perceptrons – classifying data
3. Multi-layer perceptrons – control (robotics, games)

Overview

- Learning
- What are neural networks ?
- Simple Perceptrons
- Multi-Layer Perceptrons

What will you have learnt ?

- By the end of the lecture
  - Understand what a neural network is used for
  - Understand how to create a simple network for calculating simple functions
  - Understand the basic principles behind training a complex neural network to learn from data
  - Have sufficient understanding to be able to use a Neural Network package in the tutorial to solve some problems

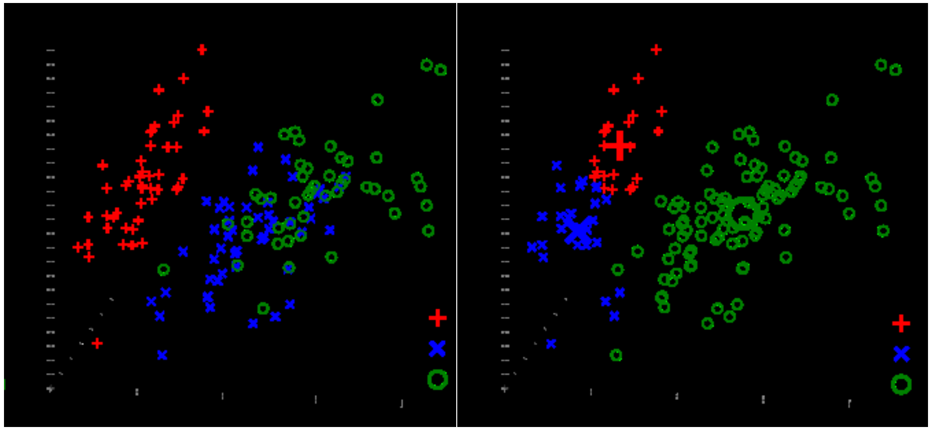
What is Learning ?

- What is *learning* ?
  - Something that improves its performance on future tasks after making observations about the world
- What is *Machine Learning* ?
  - Adaptive mechanisms that enable computers
  - to learn from experience
  - to learn by example.
  - to learn from analogy
  - to improve their performance over time

Types of Machine Learning

- **Unsupervised**
  - ask an agent to learn patterns even though no explicit feedback supplied
  - Find structure in unlabelled data
- Reinforcement Learning
- Supervised Learning

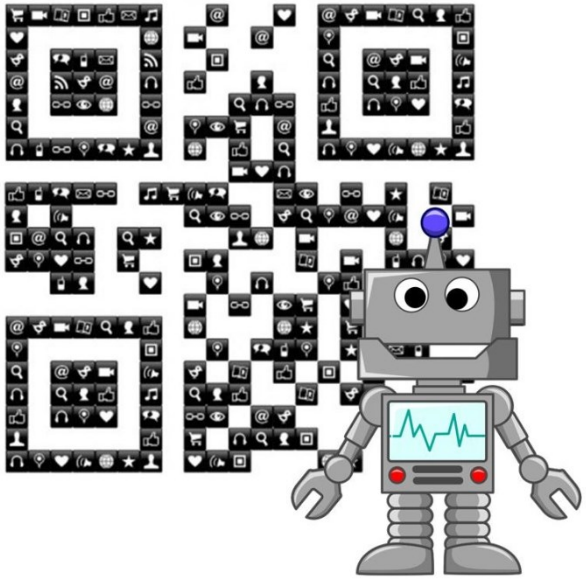
- Clustering
  - Finding potentially useful patterns in data



Types of Machine Learning

- **Unsupervised**
  - Learn to navigate a maze from the inputs to a robots sensors. The success is determined by a fitness measure(% complete or time to complete)

    (more in the third lecture)



- Reinforcement Learning
- Supervised Learning

Types of Machine Learning

- Unsupervised
- **Reinforcement Learning**
  - Agent learns from a series of reinforcements e.g rewards and punishments
- Supervised Learning

http://vimeo.com/13387420

Types of Machine Learning

- Unsupervised
- Reinforcement Learning
- **Supervised Learning**
  - *An agent observes correlations between input-output pairs and learns a function that maps input to output*
  - *Used for classification and prediction*

Car

Bus

Lorry

Some Examples

- Finance
  - Currency prediction
  - Futures prediction
  - Bond ratings
  - Business failure prediction
  - Debt risk assessment
  - Credit approval
  - Bank theft
  - Bank failure
- Medicine
  - cardiopulmonary diagnostics
- Electronic Nose
  - Senses smells and identifies the chemical

Some random applications!

1. Attach radio microphones to the top of sheep heads to transmit chewing sounds
2. Record chewing sounds and times of chewing
3. Use a neural network classifier, using your time and frequency data as input, predict future chewing periods!

More random applications!

- 1. Train a rat to press a lever, which activates a robotic arm. Robotic arm delivers reward to rat.
- 2. Attach a 16-probe array to the rat's brain that can record the activity of 30 neurons at once.
- 3. Train a neural network program to recognize brain-activity patterns during a lever press.
- 4. Neural network can predict movement from the rat's brain activity alone, so when the rat's brain activity indicates that it is about to press the lever, robotic arm moves and rewards the rat - the rat does not need to press the lever, but merely needs to "think" about doing so

A Case Study

- Recognising handwritten digits is important in many applications:
  - Sorting of mail by postcode
  - Reading forms
  - Transcribing writing from tablet computers
- NIST (National Institute for Science & Technology) in USA has archive of 60,000 labelled digits
  - Each digit is 20x20 pixels with 8-bit grayscale values

Example digits from database

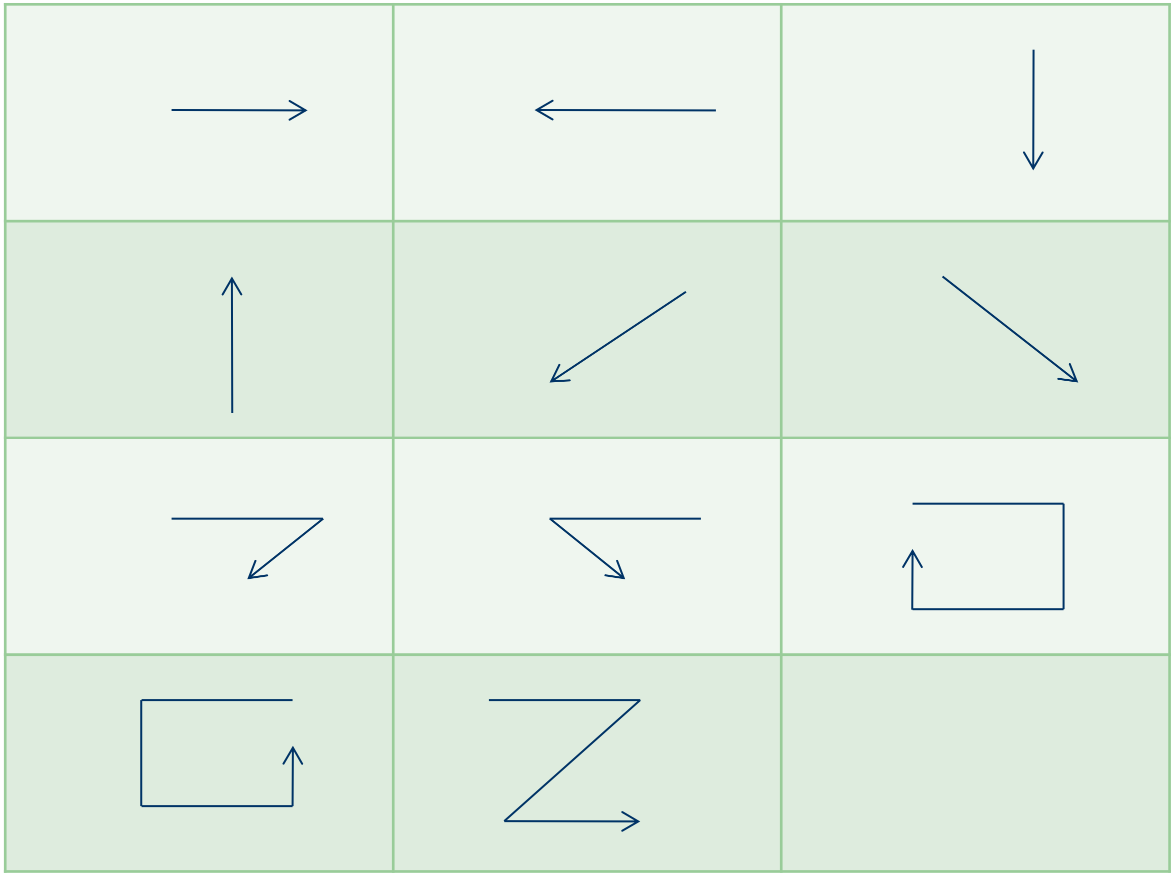
Attempts at using ML

	Error Rate	Comment
3-nearest-neighbour classifier	2.4%	No training time but slow runtime performance
Simple Neural Network	1.6%	Fast training, fast runtime performance
Specialised Neural Network	0.9%	Very long development time (over years!)
Support Vector Machine	1.1%	No training!
Humans	0.2%	But not very well tested!

Gesture Recognition:Demo

IDEA: In a computer game, easier to use mouse gestures than keyboard inputs to direct a NPC/vehicle

Train a network to recognise gestures



But some care is needed!

- In 1980, Pentagon wanted to harness technology to automatically detect camouflaged enemy tanks

- Fit tank with camera
- Camera would detect enemy tanks (e.g hiding behind trees)
- Repetitive task – hard as difficult to interpret images

Methodology

- Method:
  - Took 100 photos of trees
  - 100 photos: all with trees, 50 with tanks, 50 without tanks
  - Put 50% of each set in a locked vault
  - Trained neural net on remaining images
  - Got good results!
- Verification
  - Removed images from vault – got excellent results!
- Verification 2
  - Pentagon was suspicious
  - Commissioned 100 more photos

It didn’t work!!

What had gone wrong ?


The military was the proud owner of a multi-million dollar mainframe that could tell you if it was sunny or not!

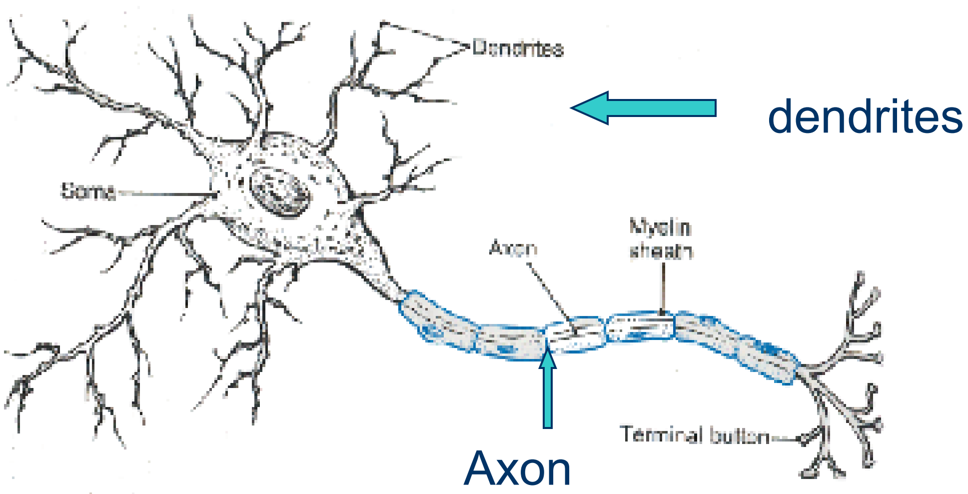
NEURAL NETWORKS

Properties of biological neural networks (the brain)

- It can learn without supervision
- It is tolerant to damage
- It can process information extremely efficiently
- It learns correlations between patterns & outputs
- It can generalise
- (it is conscious)

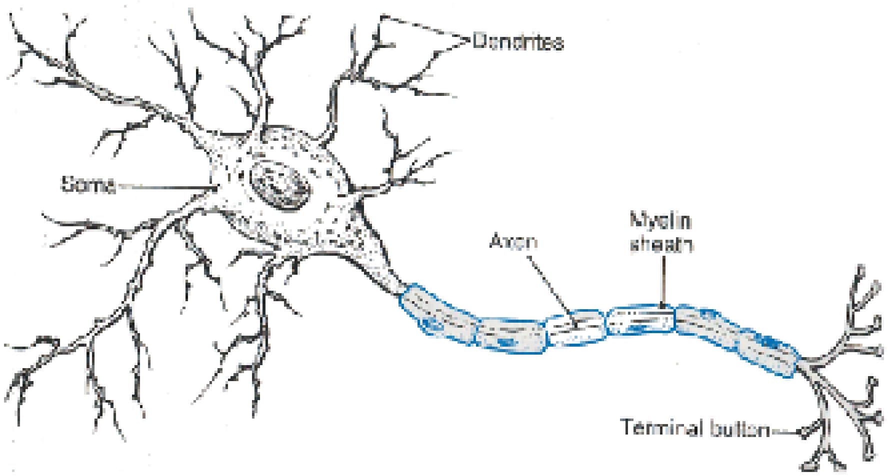
Biological Version

- Human brains contains approx. 100 billion processing units called *neurons*
  - *They are connected into a network with extraordinary processing power*



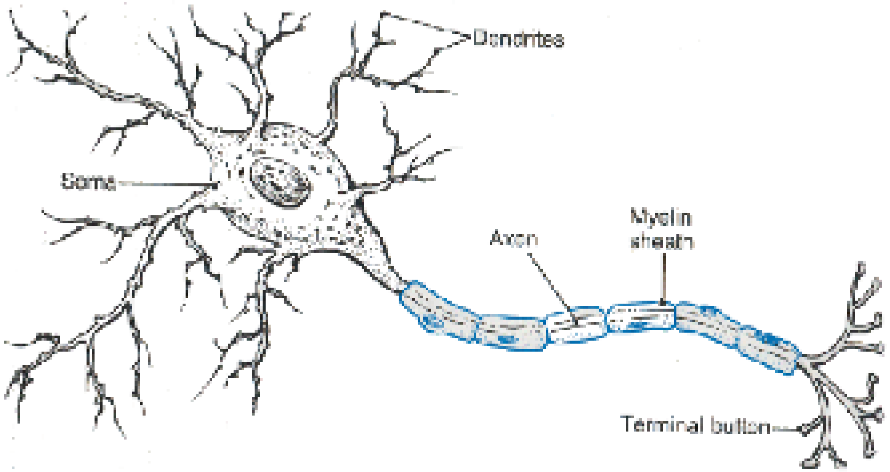
Biological Neurons

- Axon carries an action potential to other connected neurons
- The action potential is picked up by receptors in the dendrites
- Chemical reactions either inhibit or excite the potential



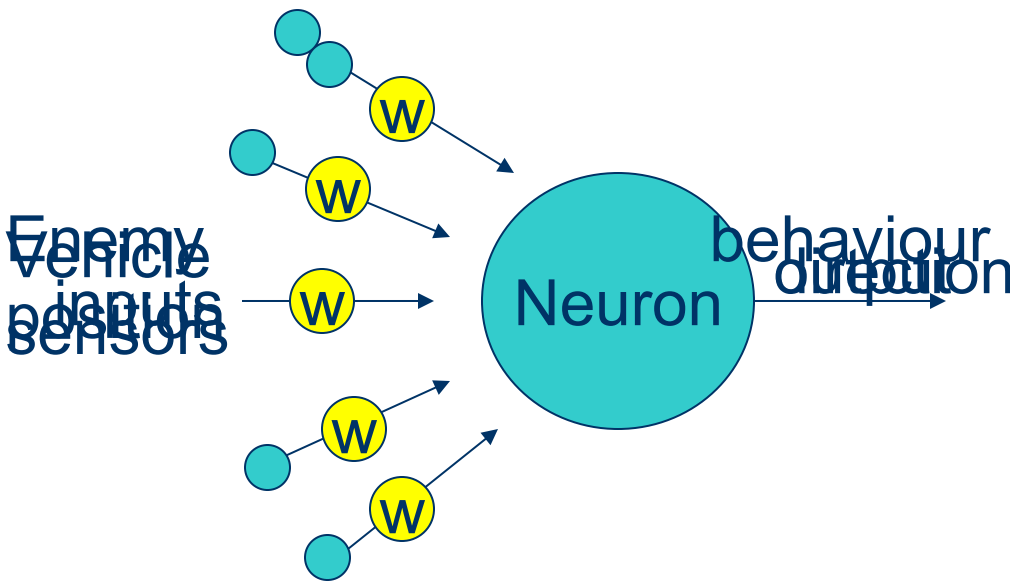
Biological Neurons

- In the brain, each neuron receives input from about 10^4 neurons
- If the combined effect of the input is sufficient it **fires**
- Firing neurons transmit their action potential to other neurons



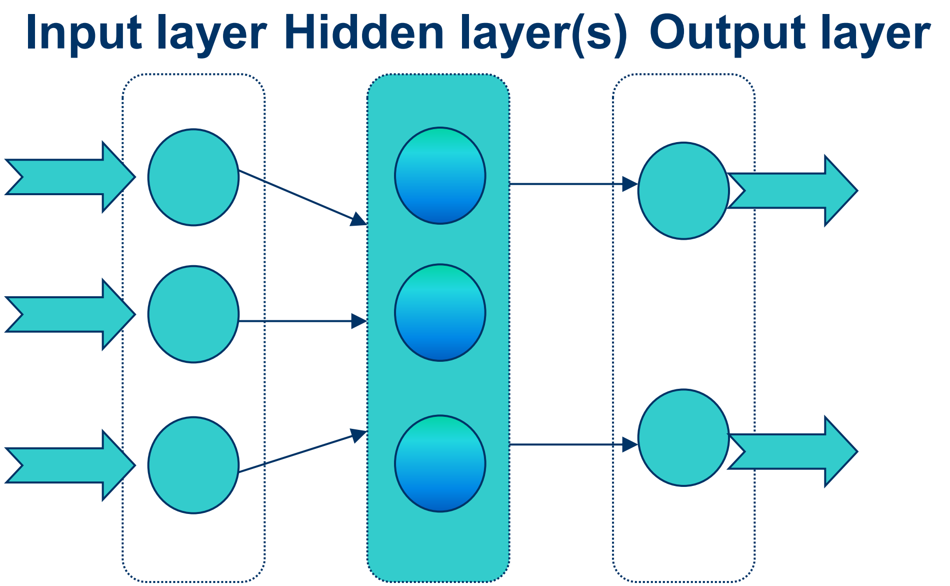
Digital Neural Networks

- Artificial Neural Networks (ANNs) are built from artificial neurons
- They have a number of **inputs**
  - which are weighted and transmitted to the neuron
- They are **trained** to produce one or more **outputs**

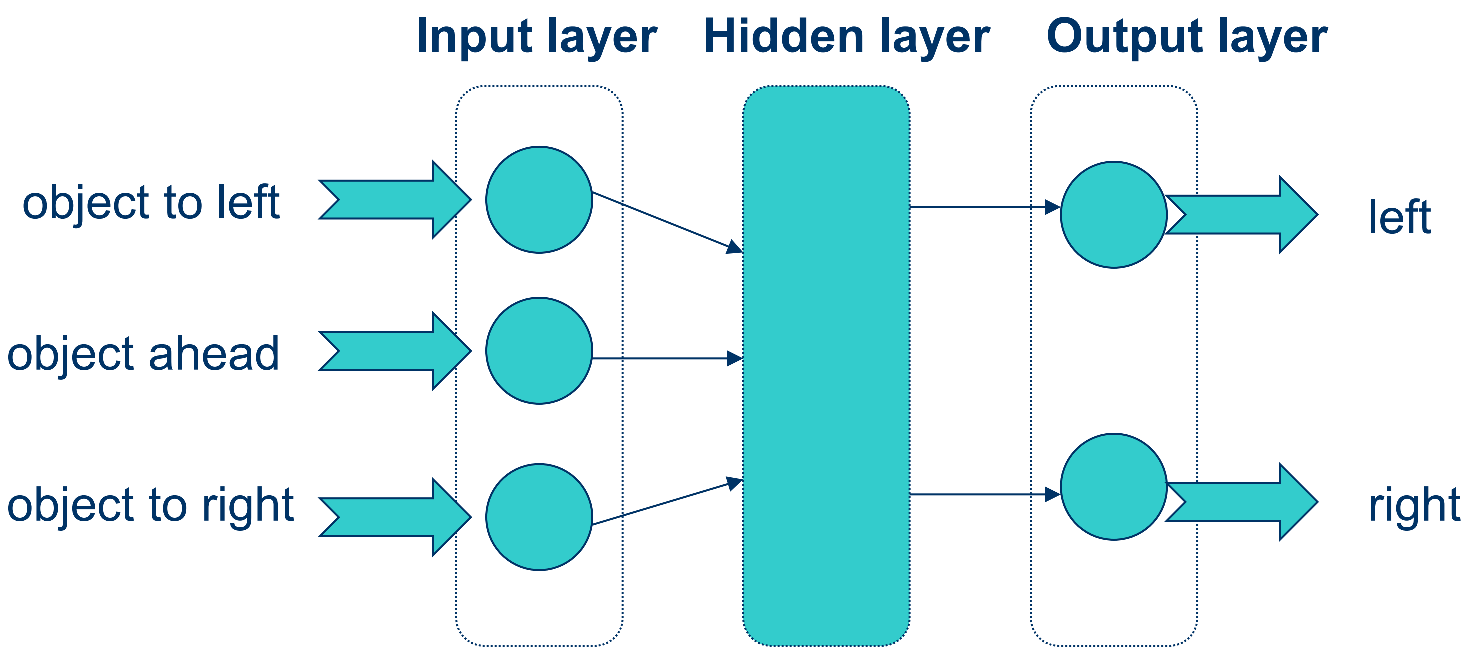


Artificial Neural Networks

- Digital neurons are combined together into neural networks
- They have:
  - Inputs
  - A black box in the middle
  - Outputs



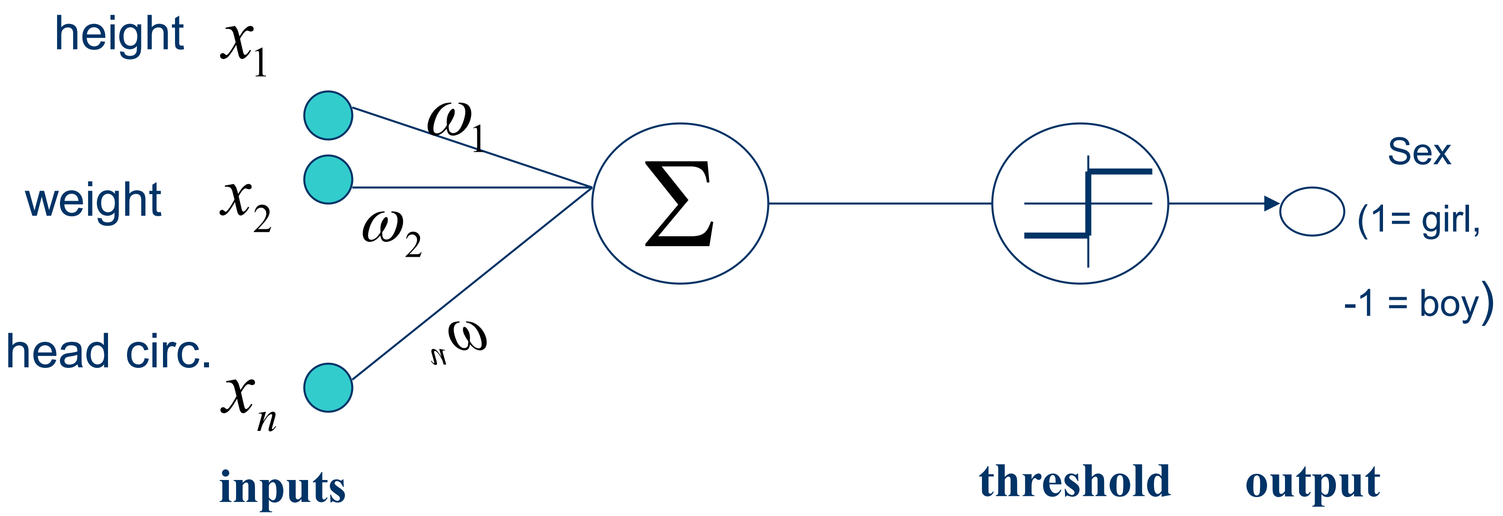
Example: Robot control



SIMPLE PERCEPTRONS

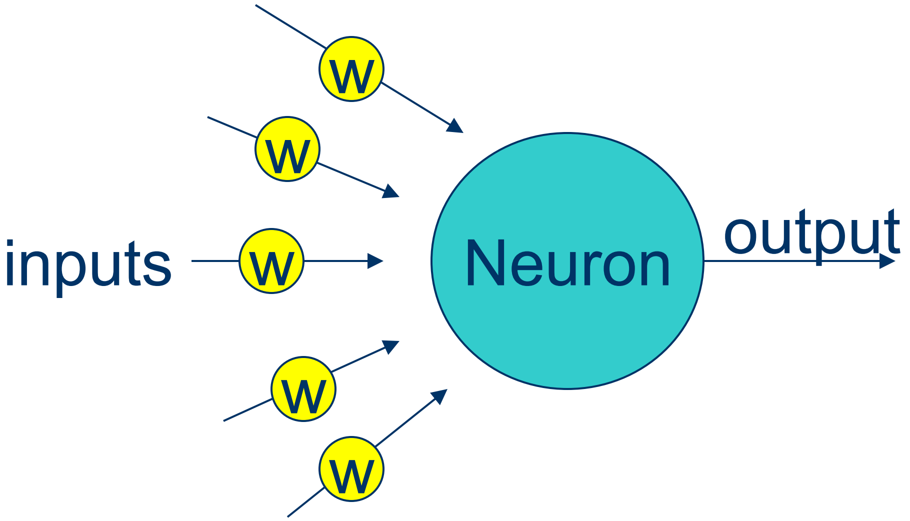
The Simple Perceptron
(McCulloch and Pitts,1943)

- Single element which takes a vector of real-valued inputs, calculates a linear combination of them, and outputs 1 if the result is greater than some threshold, or -1 (or 0) otherwise

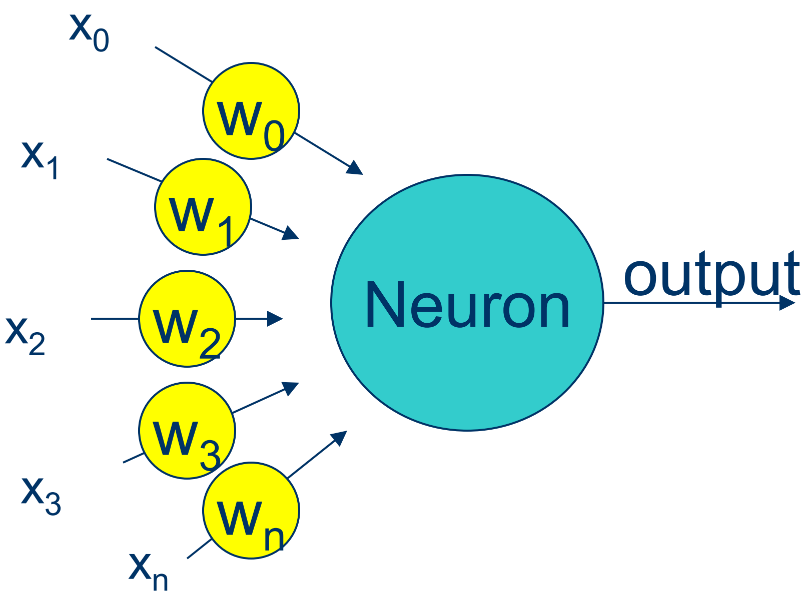


Simple Perceptron

- Each input to the neuron is multiplied by a weight
- The neuron sums the weighted inputs
- If the activation value is above a **threshold**:
  - output a 1
  - otherwise, outputs 0



Activation = weighted sum

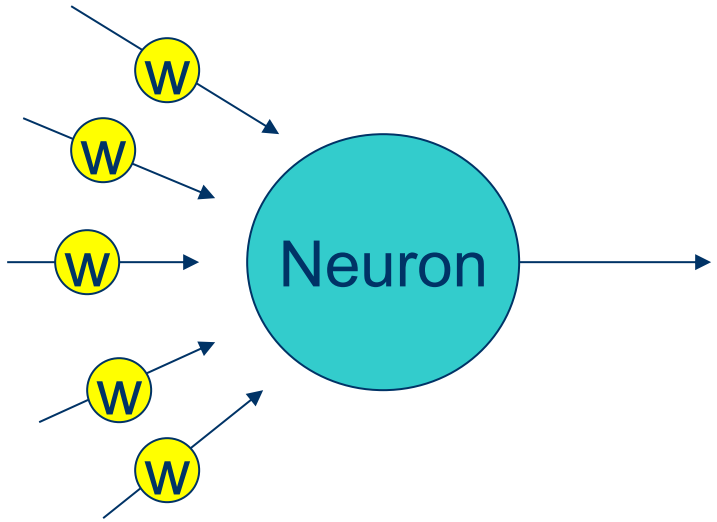


Activation = $w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4\,x_4 + w_nx_n$

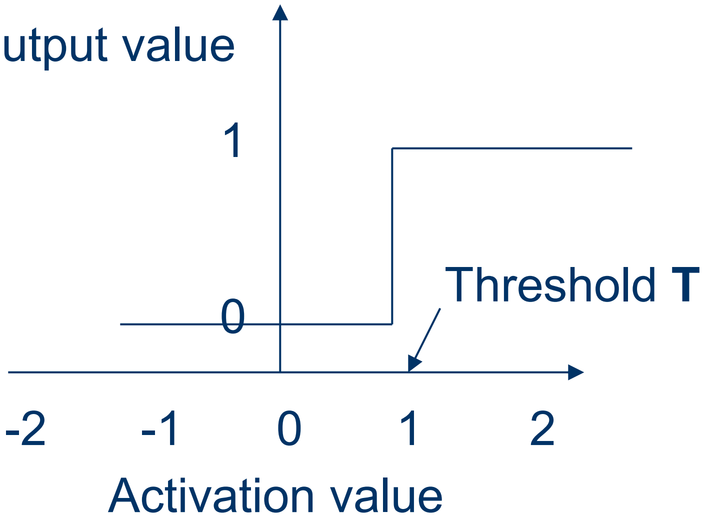
$$A = \sum w_i x_i$$

Threshold the output

if $\sum w_i x_i \geq T$ **output 1, otherwise 0**



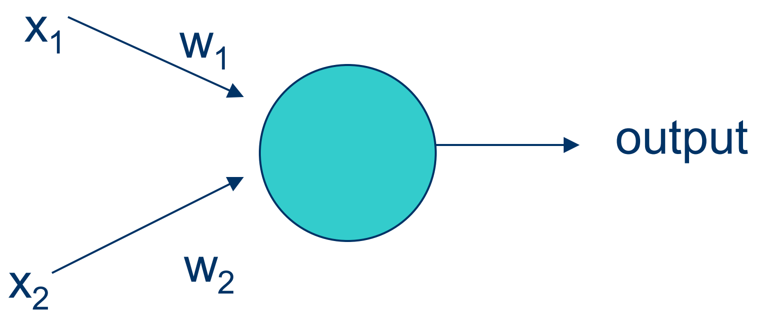
Step Function:



A Simple Example

- How should an NPC react when faced by an enemy ? 2 inputs, 1 output:

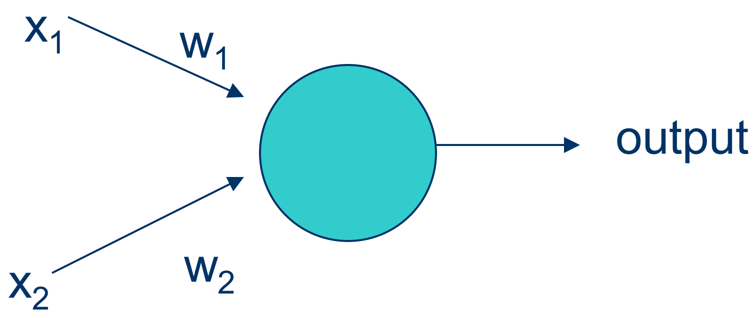
x_1	x_2	output
bullets	in-range	attack
bullets	out-range	flee
no bullets	in-range	flee
no bullets	out-range	flee



A Simple Example

- How should an NPC react when faced by an enemy ?
- 2 inputs, 1 output:

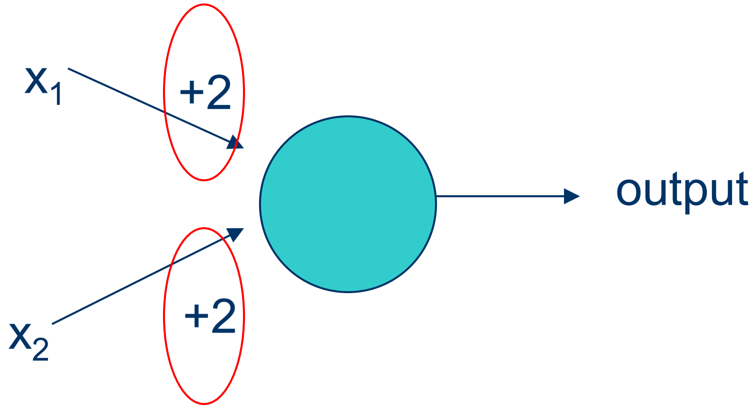
x_1	x_2	output
1	1	1
1	0	0
0	1	0
0	0	0



A Simple Example

- For each set of input-output pairs:
  - Calculate weighted sum
  - Compare to threshold to get output
- In this example, we have set (arbitrary) **threshold = 2.5** and both weights = 2

x_1	x_2	output
1	1	1
1	0	0
0	1	0
0	0	0



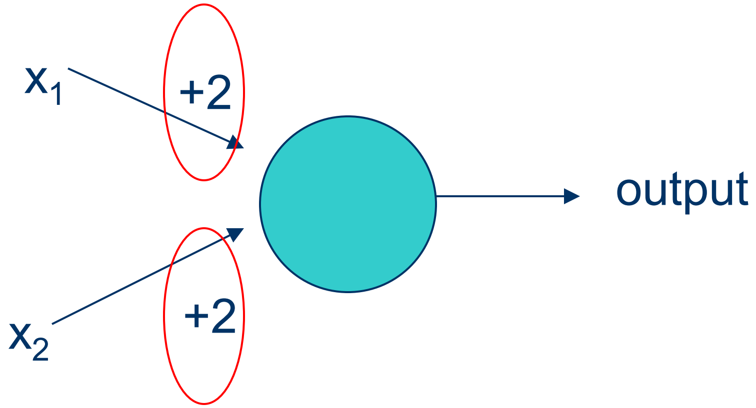
A Simple Example

Weighted sum: 1*2 + 1*2 = 4

Input 1 Input 2

If weighted sum >= threshold, output 1, otherwise 0

x_1	x_2	output
1	**1**	**1**
1	0	0
0	1	0
0	0	0

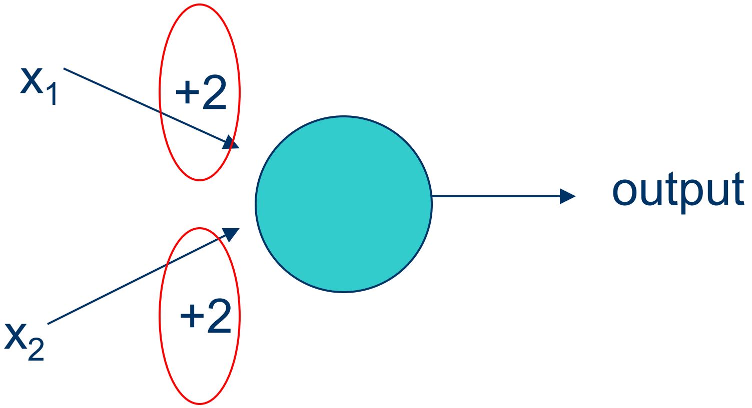


A Simple Example

- If weighted sum >= threshold, output 1, otherwise 0

1*2 + 0*2 = 2 output 0

x_1	x_2	output
1	1	1
1	**0**	**0**
0	1	0
0	0	0

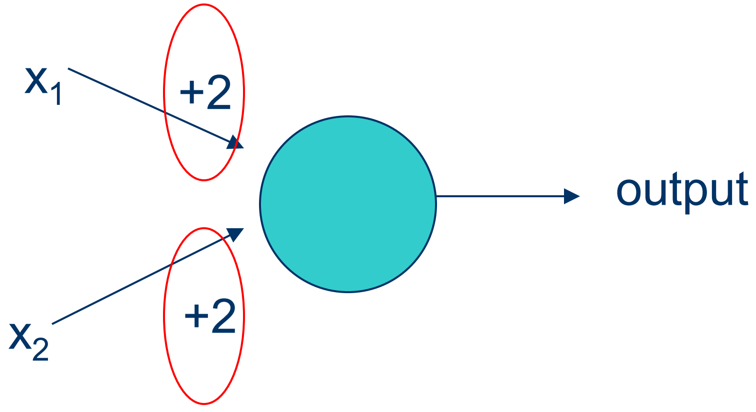


A Simple Example

- If weighted sum >= threshold, output 1, otherwise 0

$0*2 + 1*2 = 2$ output 0

x_1	x_2	output
1	1	1
1	0	0
0	**1**	**0**
0	0	0

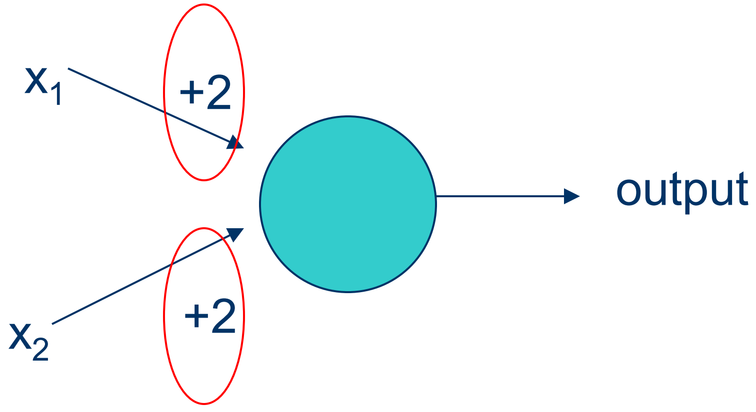


A Simple Example

- If weighted sum >= threshold, output 1, otherwise 0

$0*2 + 0*2 = 0$ output 0

x_1	x_2	output
1	1	1
1	0	0
0	1	0
0	**0**	**0**



Training

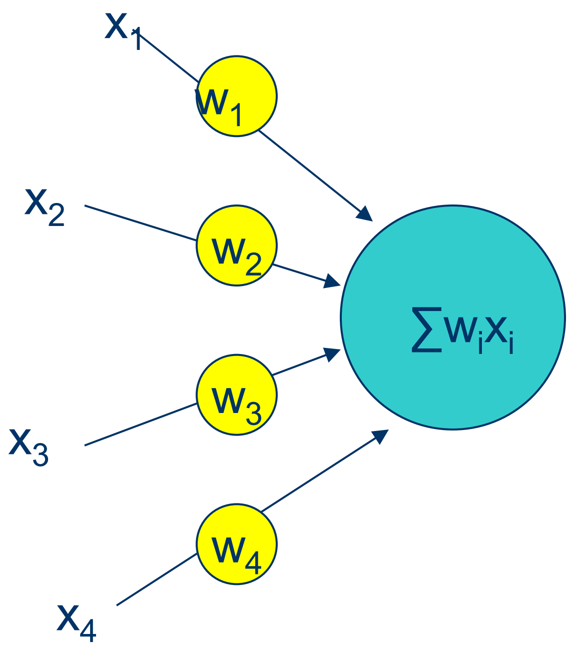
- In this example the weights and the threshold had been fixed so the network ‘works’:
  - With fixed weights and a fixed threshold it gives the correct result for all input-output pairs
- **Training** a neural network is about finding the right weights to make that network work properly for all sets of inputs
  - there is ONE fixed set of weights for all possible inputs

First let's look at that threshold

- Inputs are fed into a neuron which sums the weighted input
- Neuron fires if activation greater than a threshold:

$$x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4 > T$$

- **But how do we set T ?**
  - If we need to find weights, why not find T as well ?



Thresholds

- We can rewrite the equation;

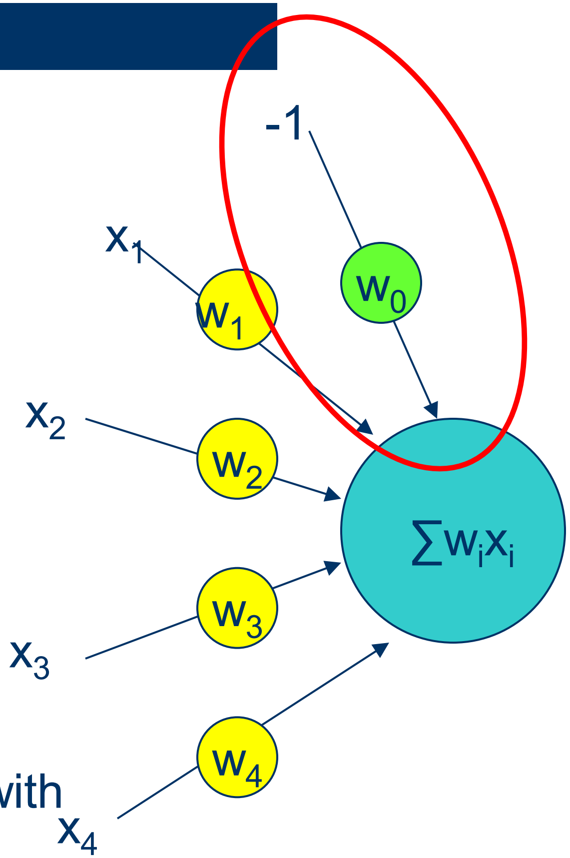
$$x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4 > T$$

$$x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4 - T > 0$$

$$x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4 + (-1)\ T > 0$$

Looks like an **input**, with value -1

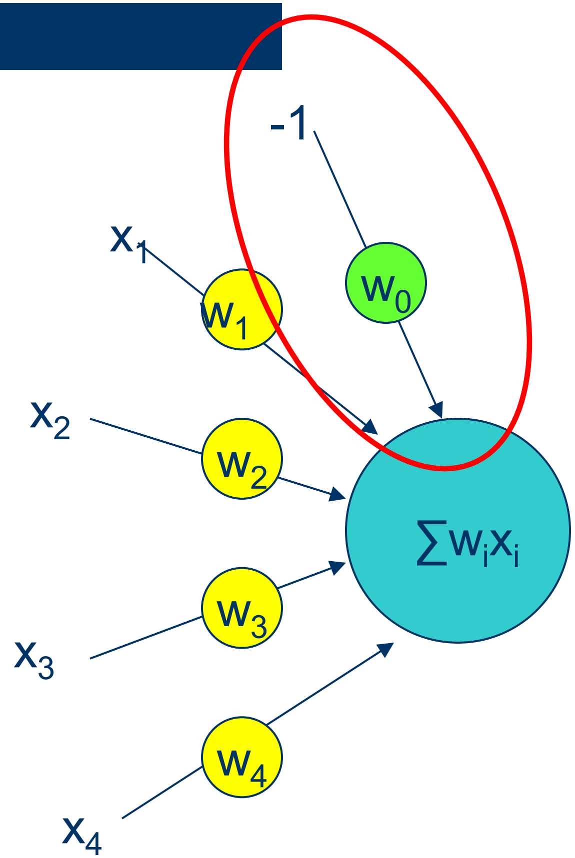
Looks like a **weight** with value T



$$x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4 + x_0w_0 > 0$$

Removing the threshold

- We can ignore the threshold by adding an extra input to those required by the application
- The extra input is called the **bias**
- *It always has input **value -1**
- It has a **weight w_0** which needs to be calculated



that in practice, most software assumes the input is actually +1 and the weight is –w (which achieves the same effect)

Re-written output rule

- By incorporating T into the weights, we can re-write the output rule:

if	$$\sum_{i=0} w_i x_i \geq 0$$	**output 1**

if	$$\sum_{i=0} w_i x_i < 0$$	**output 0**

Finding weights

- By including the bias we don't have to worry about the threshold any more
- But we still need to find a method of calculating the correct values for the weights
- For single perceptrons, we can do this with the **Perceptron Learning Rule**

But how do we learn what the weights should be?

1. Start with a random choice of weights
2. Repeat for n iterations:
   a) Present a sample from the training set:
      1. if a neuron outputs 0 when it should output 1, *increase* each weight a bit
      2. if a neuron outputs 1 when it should output 0, *decrease* the weights a bit
      3. Go back to a) until all input patterns presented

This is the Perceptron Learning Rule

Perceptron Learning Rule

1. Initialize the weights and threshold to small random numbers.
2. Present a vector **x** to the neuron inputs and calculate the output.
3. Update the weights according to:
4. Repeat steps 2 and 3 until:
   - the iteration error is less than a user-specified error threshold
   - or predetermined number of iterations have been completed.

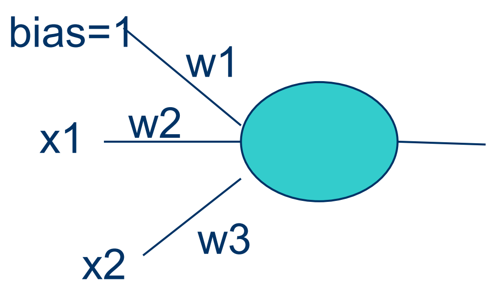
desired actual

$$w_j(t+1)= w_j(t)+\eta(d-a)x$$

Learning rate input

Example

- Assume weights are <-1, 2,-3> and learning rate is 1 (so we can ignore it)
- Present input pattern <1,1>
- Calculate output $\sum_{i=0} w_i x_i$
  - -1*1+2*1+-3*1
  - -1+2+(-3) = -2
  - Sum < 0, output 0
- Incorrect: apply update rule:
  - w1: -1 + (1-0) * (bias=1) = 0
  - w2: 2 + (1-0) * x1 = 3
  - w3: -3+(1-0) * x2= -1
  - <0,3,-1>



x1	x2	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

$$w_j(t+1)= w_j(t)+\eta(d-a)x$$

Learning Rule

- The Perceptron Learning rule can be proved to converge after repeated iterations (*as long as the data is **linearly separable***)
- Can be inefficient
  - Imagine a dataset with 100 records
  - 99 records give the right output
  - When you present record 100, the output is wrong
    - Forces weights to change
    - New weights now make (some) of the previous 99 records wrong!

Voted Perceptron

- Like the learning algorithm just shown, except:
- **Each time you make a weight update due to an incorrect record:**
    - **save the current set of weights <w>**
    - **Record the number of records that were presented during the that this set of weights remained unchanged (v)**
- After a fixed number of iterations, there will be a (long) list containing all the saved sets of weights
- Load each set of weights into a new perceptron
- For each data record, each of the new perceptrons **votes** on what it thinks the output is
- Votes are weighted by v, the number of times a weight set survived
- The output class is determined as the one with most votes

Voted Perceptron

R1
R2
R3

<w1> : 3 records

R4
R5
R6
R7
R8
R9
R10
R1
R2
R3
R4
R5
R6
R7
R8

<w2> : 15 records

R9
R10
R1
R2
R3
R4
R5
R6

<w3> : 8 records

R7

<w4> : 1 records

R8
R9
R10
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10

<w5> : 12 records

Voted Perceptron

- 5 sets of weights stored – create 5 perceptrons, one with each set
- Present record 1:
  - P1 outputs 1
  - P2 outputs 1
  - P3 outputs 0
  - P4 outputs 0
  - P5 outputs 1
- Votes for class 1:
  - P1+P2+P5 = 3 + 15 + 12 = 30
- Votes for class 0:
  - P3+P4 = 1+8 = 9
- Assigned class = 1
- Repeat with each record..

<w1> 3

<w2> 15

<w3> 8

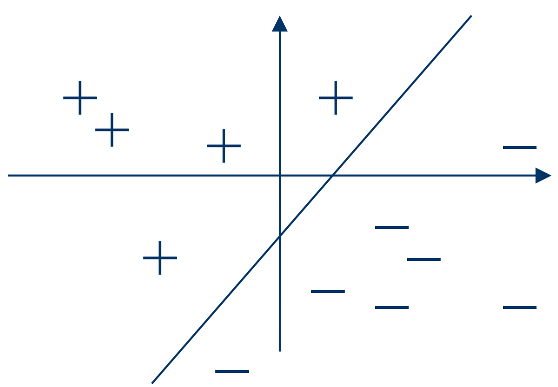
<W4> 1

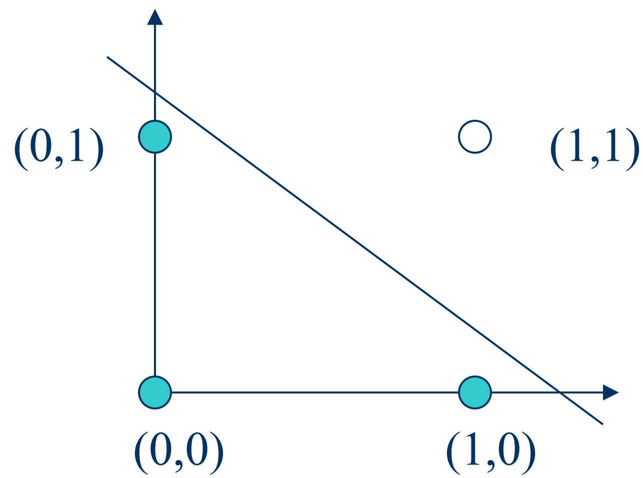
<W5> 12

WHAT CAN PERCEPTRONS LEARN ?

What can be learned ?
A geometric point of view:

- View perceptron as representing a *hyper-plane decision surface* in the n-dimensional input space
- Perceptron outputs 1 for points on one side of the surface, and 0 for points on the other

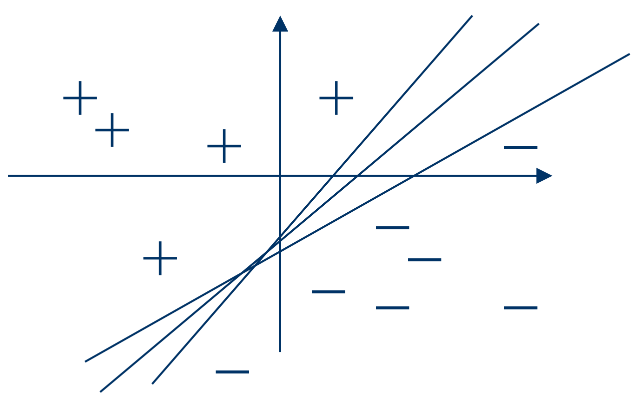


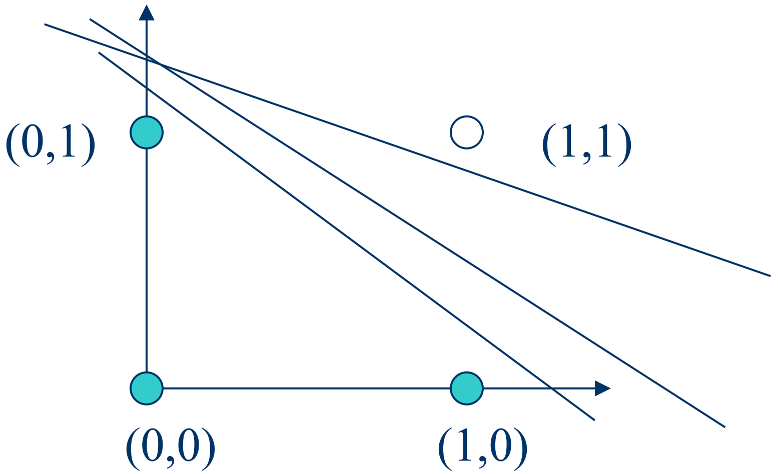


AND gate

What can be learned ?
A geometric point of view:

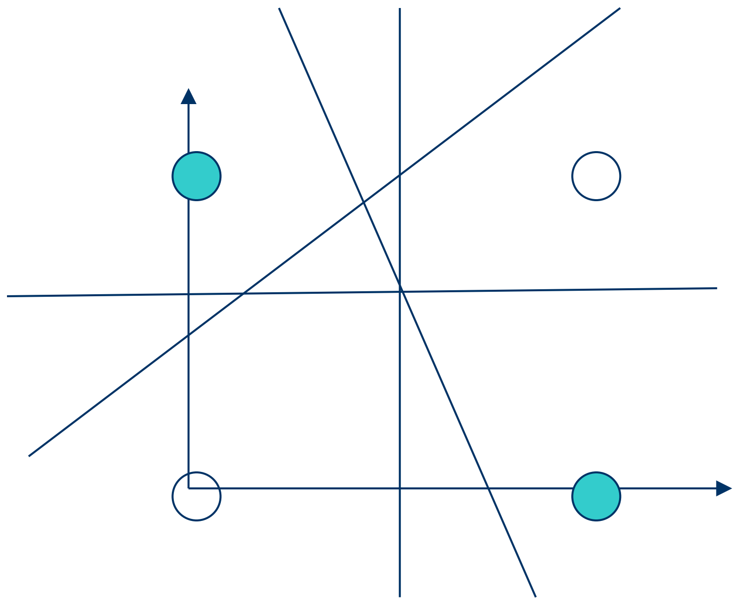
- A perceptron can learn to classify any data where it is possible to separate the two classes with a line..
- Any line that separates them is OK
- The perceptron weights effectively define the equation of the line .. So there are many sets of weights that provide a correct solution





AND gate

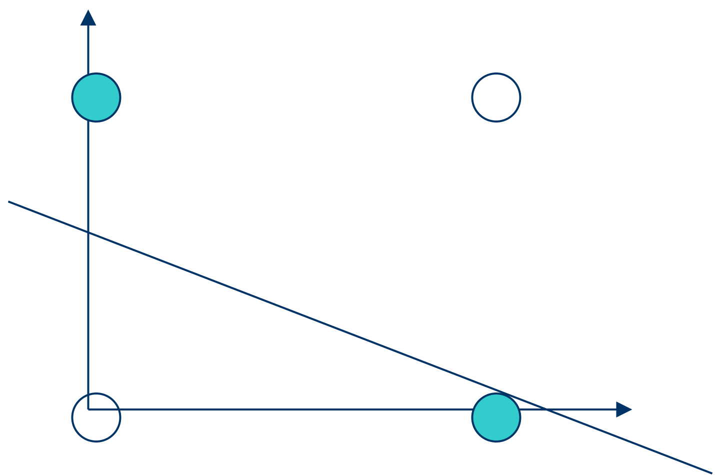
Some things can't be learned though....



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

- Consider XOR

Some things can’t be learned though....



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

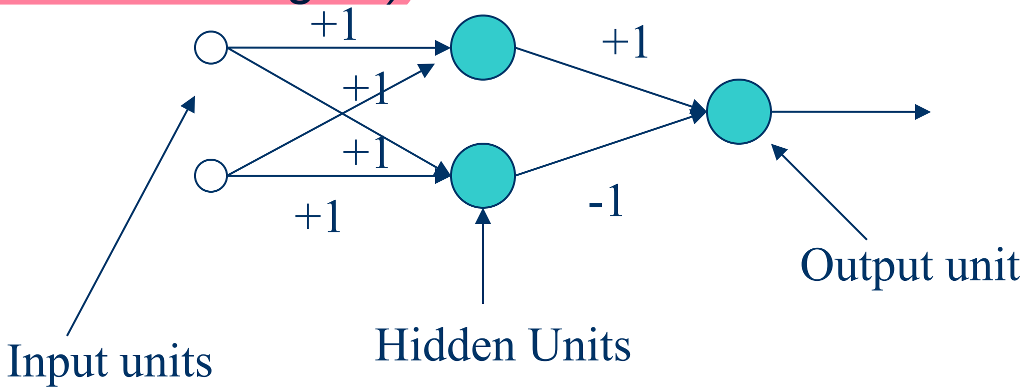
In order for a perceptron to be able to learn, the data must be **linearly separable,** i.e. you can separate points of two classes by a straight line (or surface in higher dimensions)

Multilayer perceptrons

- Single perceptrons can be combined into networks with a ‘hidden layer’ of units between the inputs and outputs
- Any *Boolean function* can be represented by some combination of perceptrons
- (*the problem is how to find the weights*)

A network that

can do XOR



Multilayer Perceptron

- Next week we will look at how to combine perceptrons into an MLP to classify any data set
- We will look at how to train an MLP
  - The Perceptron Learning Algorithm can't be used
- We will look at how to prepare data for using with a neural network
- … and how to evaluate its performance

Summary

- We have learned:
  - How a simple perceptron works
  - How to train a simple perceptron
  - The limitations of simple perceptrons
  - That combining perceptrons into a multi-layer network can in theory be able to represent all Boolean functions

Tutorial

- Creating neural networks
- We will use some software called WEKA
  - Can be used from GUI or as a java library
- You will create some networks to solve logic functions (AND, OR)
- We will show that XOR can't be solved
- We will create more complex data sets to explore this concept further
- Some exercises relating to linear separability