Artificial Neural Networks

Artificial Neural Networks

- **Introduction**
- **Machine Learning**
- History
- Creating/Training ANNs
- Data processing
- Evaluation
- A few things to finish

Why Artificial Neural Networks?

- Consider pattern recognition
- Recognising images in the rain
- Difficult AI task

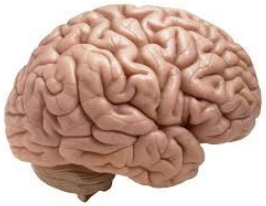
1) Low level image processing to identify line segments and regions

2) Deduce shadings etc

3) Construct a 3D object in a semantic structure

Machine Learning

- We perform these actions without a second thought.

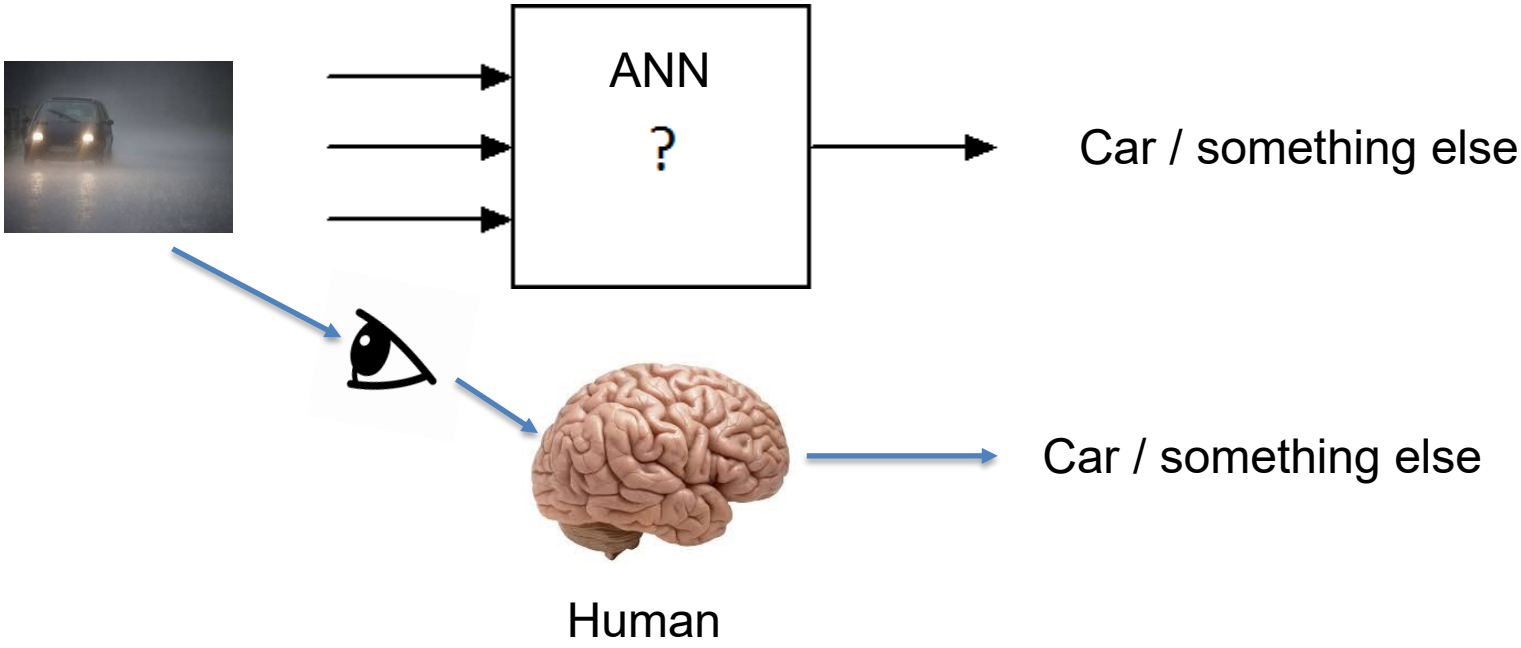


- *Machine learning* may give best hope for being able to perform difficult tasks.

- Machine learning – computer models that improve their performance based on data.

Machine Learning

- Black box interpretation – difficult task



Machine Learning – example difficult task

BBC Sign in Home News Sport Weather iPlayer Sounds

NEWS

Home | Coronavirus | Climate | UK | World | Business | Politics | Tech | Science | Health | Family & Education

Technology

Supermarket cameras to guess age of alcohol buyers

By Jane Wakefield
Technology reporter

2 hours ago

The system can work out if a face is older or younger than 25

Major supermarket chains have begun testing an automated age-verification system, to avoid the wait for staff at self-checkouts when buying alcohol.

The trial will use cameras that can estimate each customer's age.

[BBC News 2 Feb 2022]

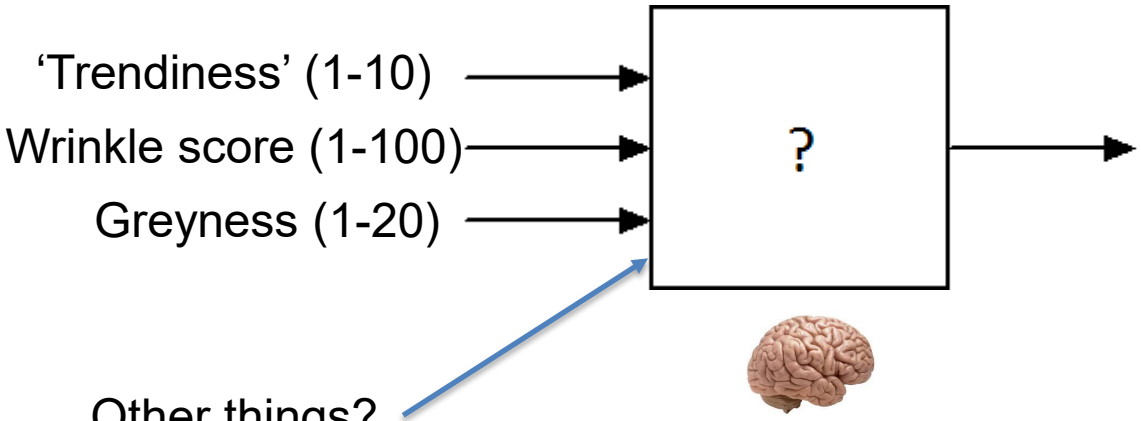
Loughborough University

Machine Learning – example difficult task

Estimating a person's age

Our data set (source?)

	Predictors			Predictand
	Trendiness	Wrinkles	Greyness	Age
Person 1	5	45	2	22
Person 2	9	23	1	19
Person 3	3	72	18	38
etc	etc	etc	etc	etc



Jeanne Calment

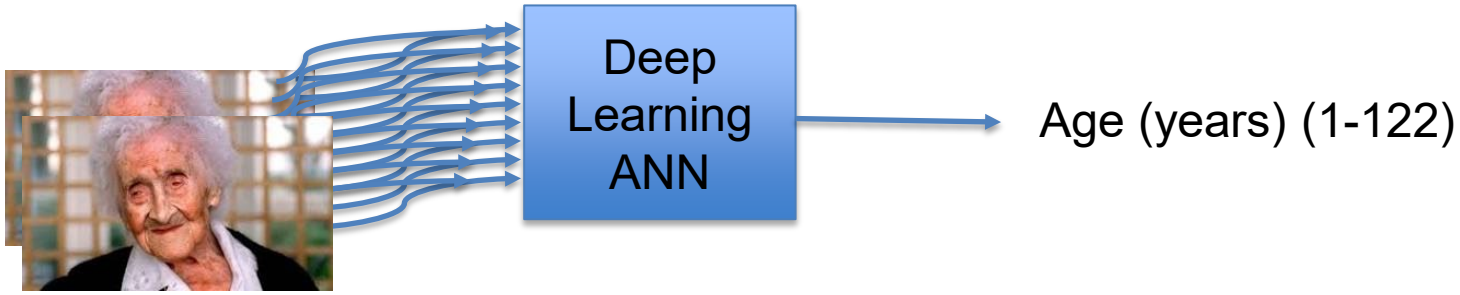
(35)

Loughborough University

Machine Learning – example difficult task

Estimating a person's age

Deep Learning ANN - input an image – so loads more inputs



Eg 10 Megapixel image ~ 10 million inputs
Each pixel – could be 24-bit = 2^{24} ~ 17 million colours

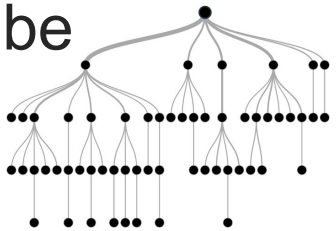
Also – what if we move image slightly, would lead to a different set of inputs?

Machine learning

- Inducing general functions from training examples.
- Inductive learning – discovering rules/relationships by observing examples
- No universal algorithm that can take a sample of training examples from an arbitrary unknown function and produce a good (perfect) approximation to that function.
- Each algorithm searches the hypothesis space and each uses a different structure to organise the search through the hypothesis space.

Machine Learning Algorithms - Examples

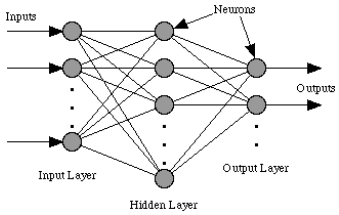
- **See5 algorithm** - assumes unknown function can be represented by a decision tree.



- **Radial basis function network** - models the unknown function using a network of Gaussian basis functions.

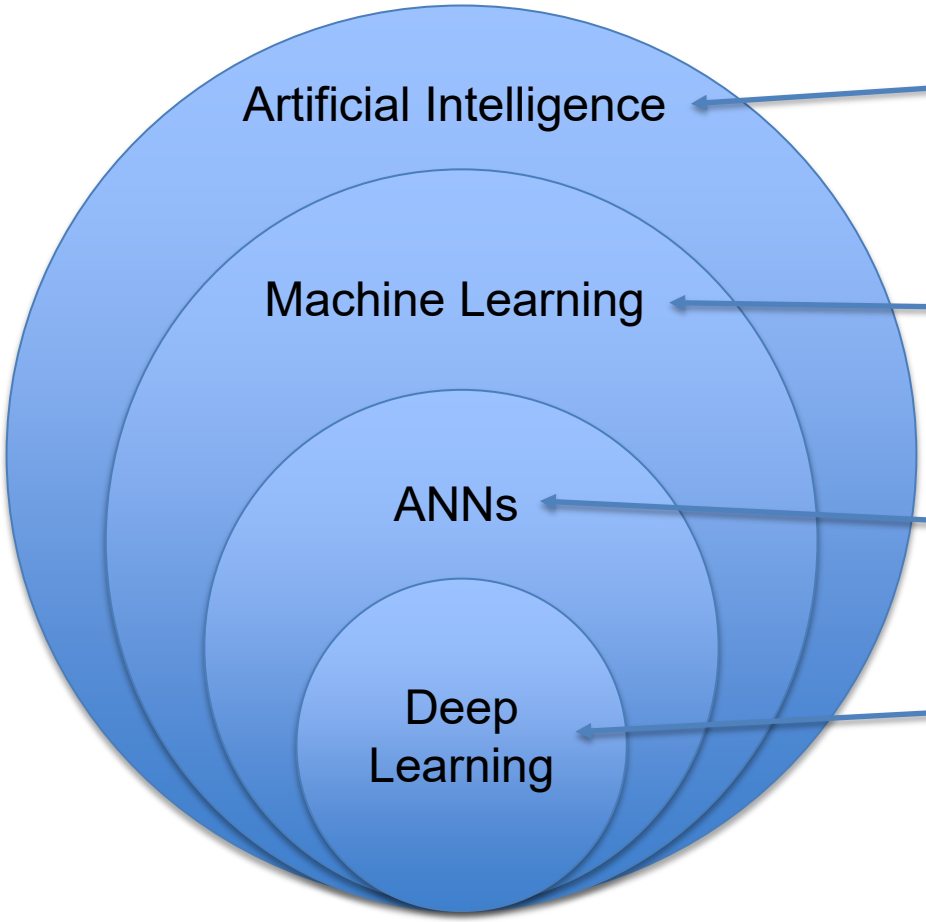
- **Multi layer perceptron** (MLP) - assumes unknown function can be represented by a multilayer, feed forward network of sigmoid units.

(Aka an ANN)



Our focus

AI v ML v ANNs v DL



Techniques which enable machines to mimic human behaviour

Subset of AI in which statistical methods are used to enable machines to improve with experience

Machine Learning approach using neural networks

ANNs with many different layers

[Adapted from : https://www.edureka.co]

Machine Learning

- **Supervised** – teacher indicates how well model is performing during training phase
- **Unsupervised** – no performance evaluation available – most we can expect is system to group data into similar input patterns – clustering.

Supervised v Unsupervised - a cricket analogy!

- I want to teach my son (Jacob) cricket – batting
- I can't get all the world's bowlers to bowl at him
- I get a bowling machine with 100 different settings
- The machine fires 100 different types of ball at my son – these are our data set
- The balls could be random or in a sequence (think of issues with this)

Supervised v Unsupervised - a cricket analogy!

Supervised learning

- I get the machine to fire each 100 balls at him and feedback to him how well he is doing after each ball
- Repeat – cycling through the 100 balls again and again (how many times?) until he gets better and better at dealing with them all
- Downside – I 'overtrain' him on these 100 balls – so when he plays in real life, and encounters a different ball (not seen before), he does not know how to deal with it

We will look at ways to avoid over-training ANNs later

Supervised v Unsupervised - a cricket analogy!

Unsupervised learning

- I get the machine to fire each 100 balls at him but don't tell him how to bat or how to deal with them
- Repeat – cycling through the 100 balls again and again – he learns his own way of hitting the ball
- I then observe how he plays – hopefully learning some new strokes myself or new ways of dealing with different balls bowled that I had never thought of before
- I have 'mined' some data from an existing data set – data mining – using an unsupervised ANN model

Supervised v Unsupervised - a cricket analogy!

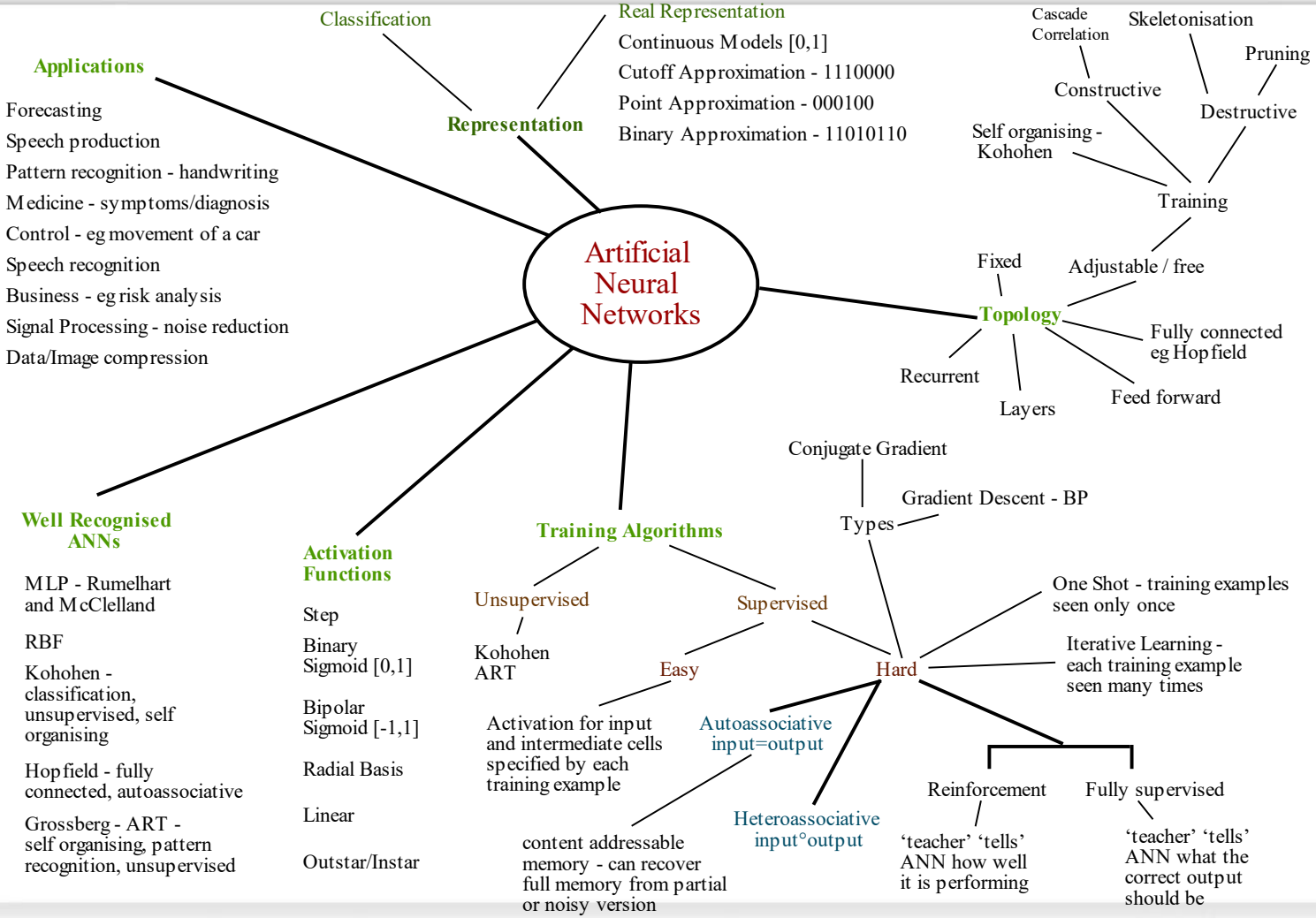
Remember - Machine learning – computer models that improve their performance based on data.

The more data I can get, or better ‘quality’ data, the ‘better’ I can make my model

If I had a more expensive bowling machine with 1,000 settings, I could train Jacob to perform even better and deal with a wider variety of bowlers in the real world (although training would take longer).

So – our ANN models are very much dependent on the data we have available.

ANNs – a possible research territory map

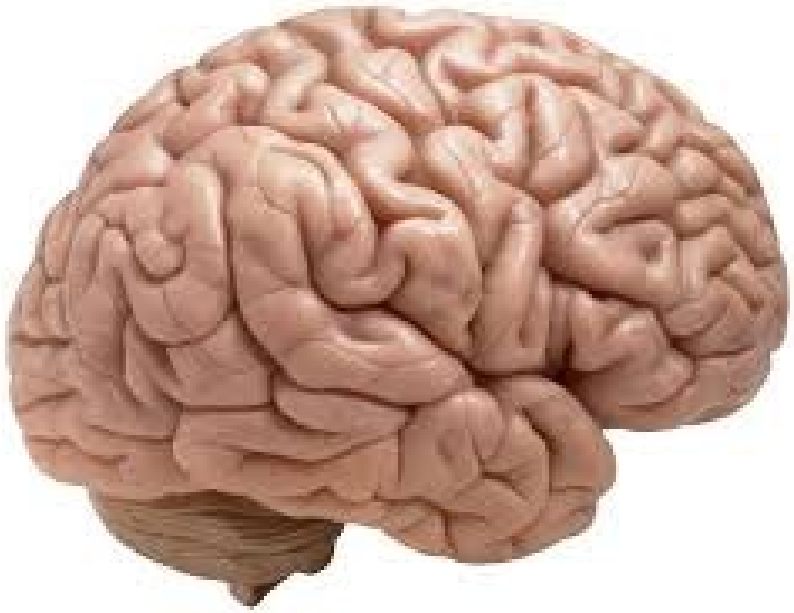


The human brain

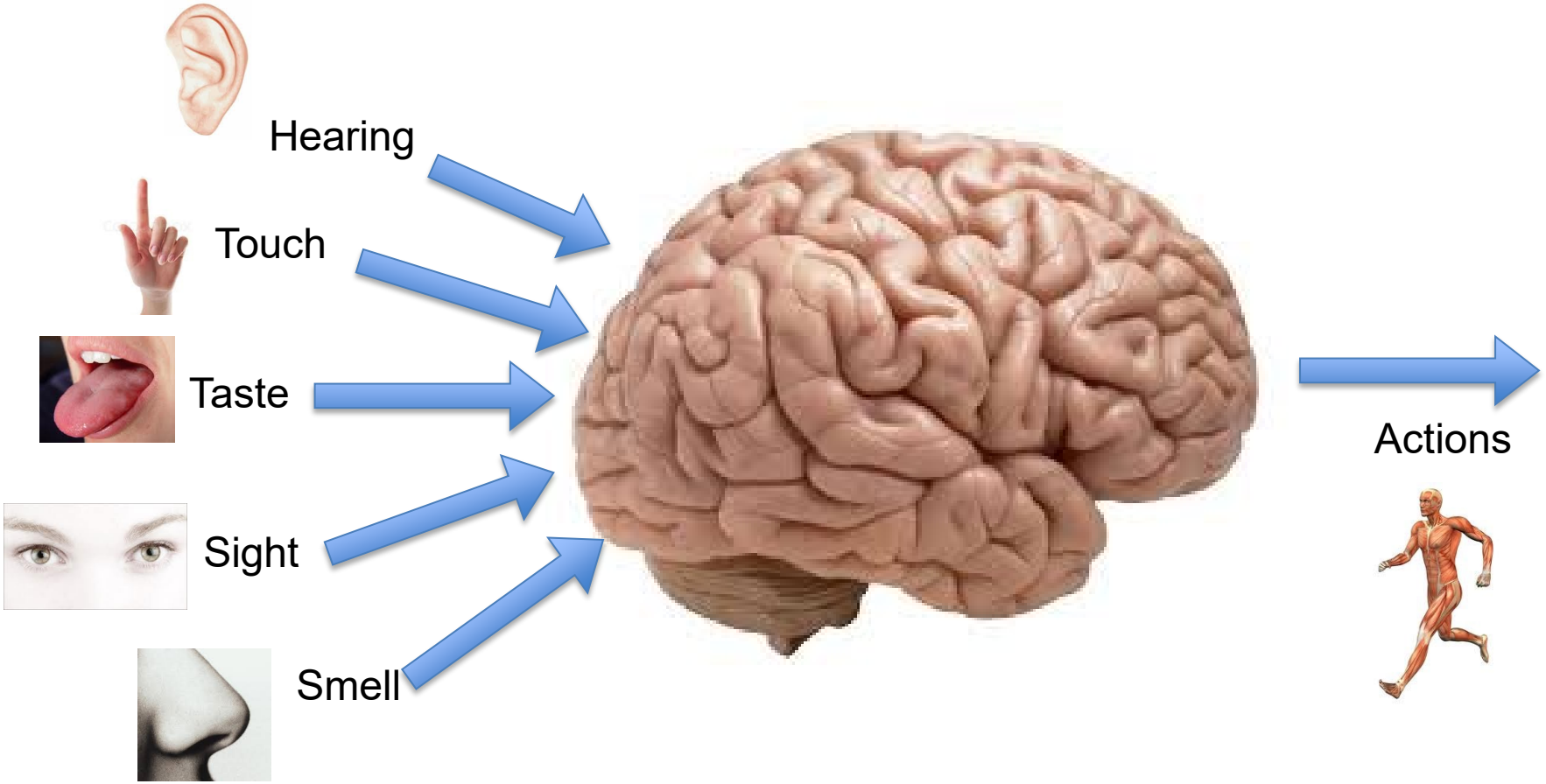
- 10^{11} brain cells (100 billion)
- Each connected to 7,000 others
- 10 x world population

(7 billion)

x 10 =

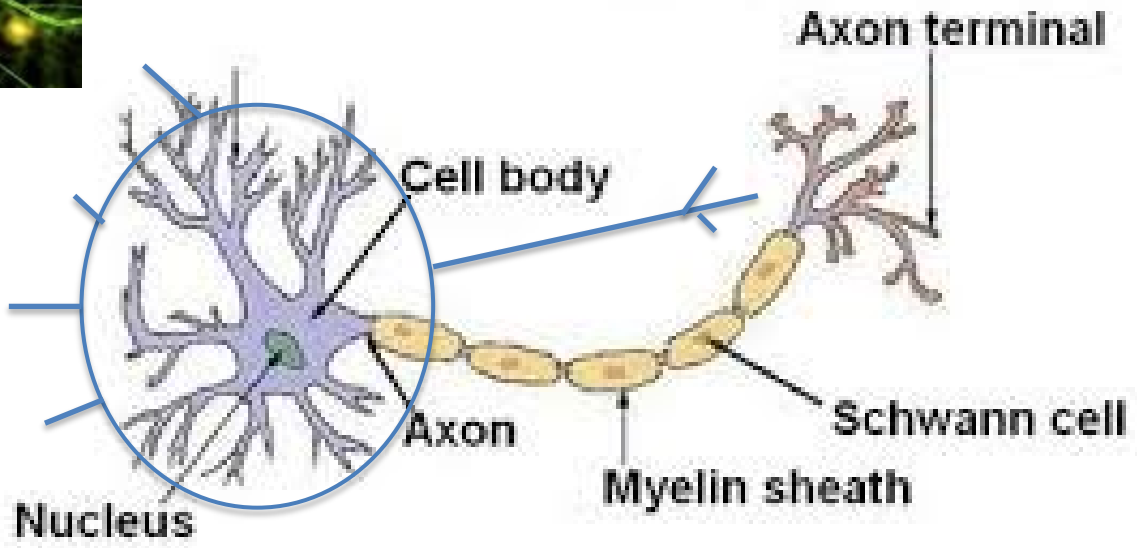


The human brain - learning



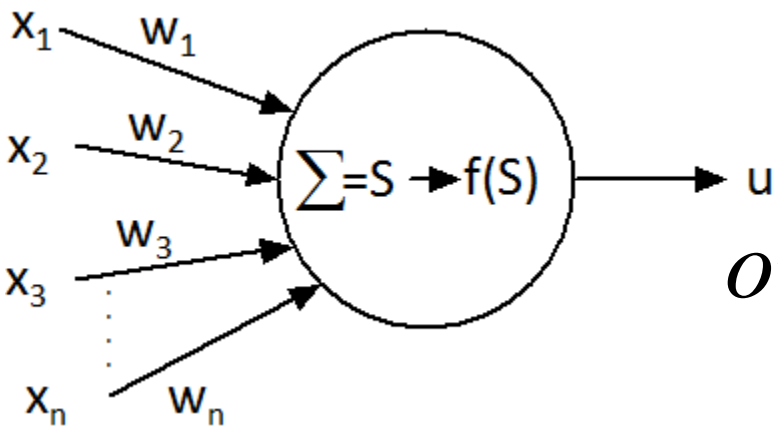
Brain Cell

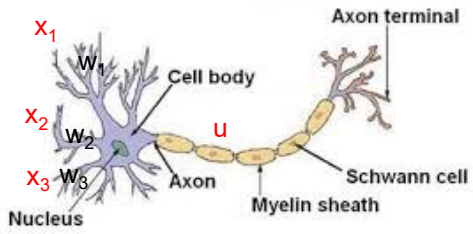
(10^{11} brain cells)



The artificial neuron – single cell

Output is a function of the weighted sum of the inputs





$$output = u = f(S)$$

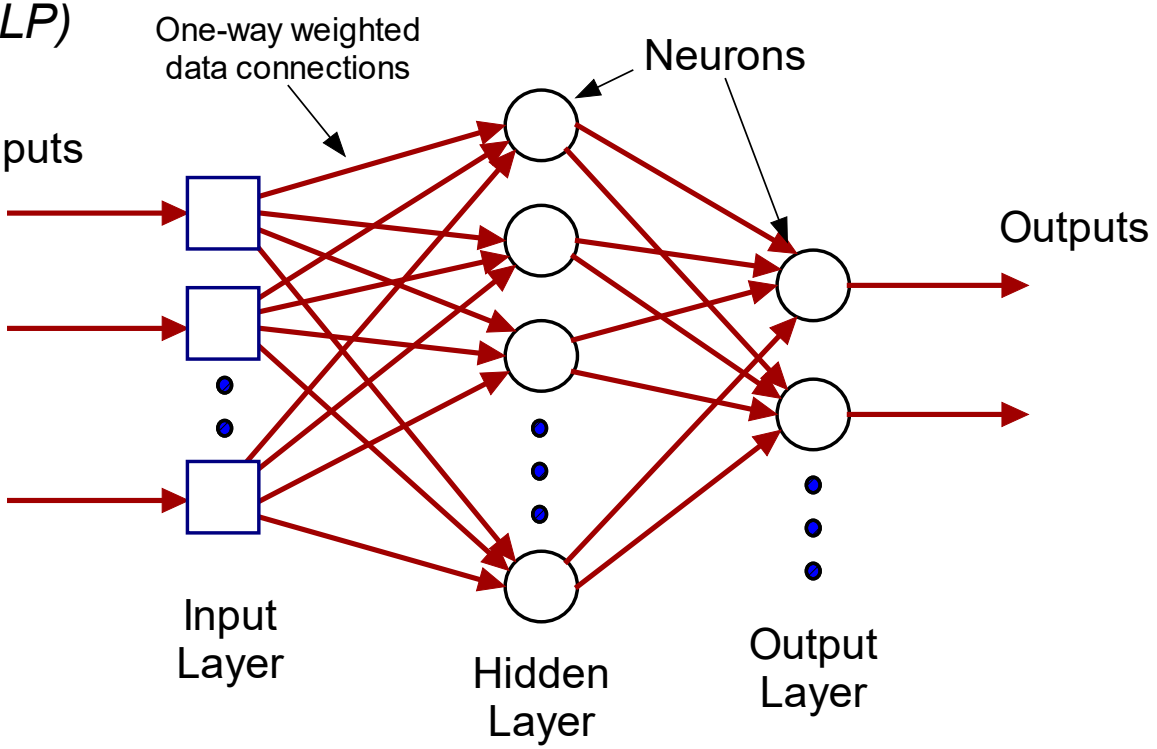
$$S = x_1w_1 + x_2w_2 + x_3w_3 + \ldots + x_nw_n \qquad S = \sum_{j=0}^{n} w_j x_j$$

$f(S)$ is a suitable function - more on this later

(aka. Activation Function; Transfer Function)

An Artificial Neural Network – feed forward

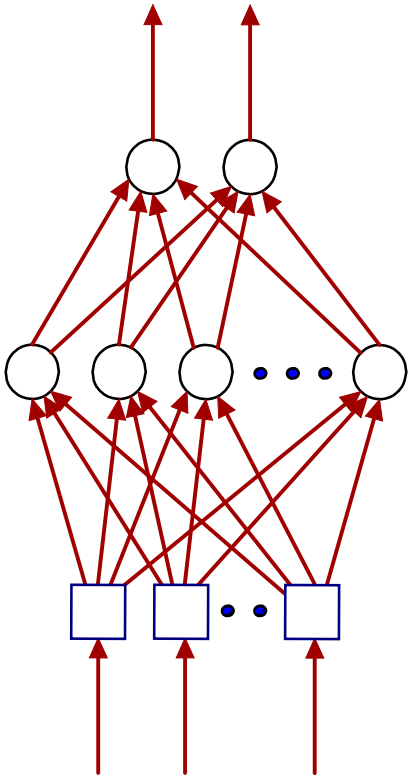
Neural Network
Artificial Neural Network (ANN)
Connectionist Network
Multi Layer Perceptron (MLP)



Nodes
Neurons
Cells
Perceptrons
Computational Units

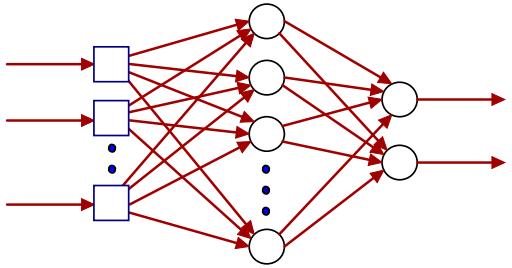
(could have many hidden layers)

ANN - feedforward

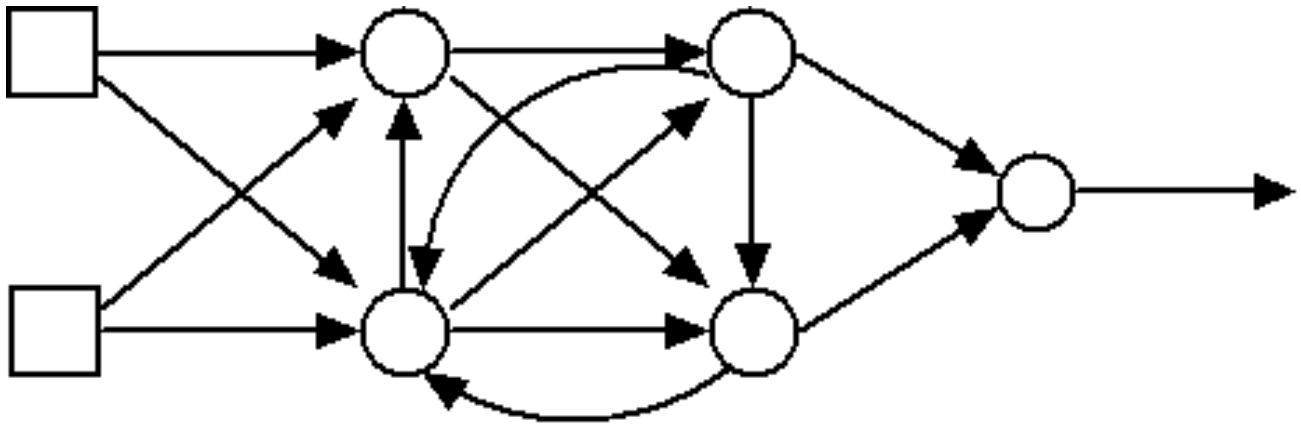


Some books draw the ANN upwards

I prefer left to right:



Recurrent / feedback network

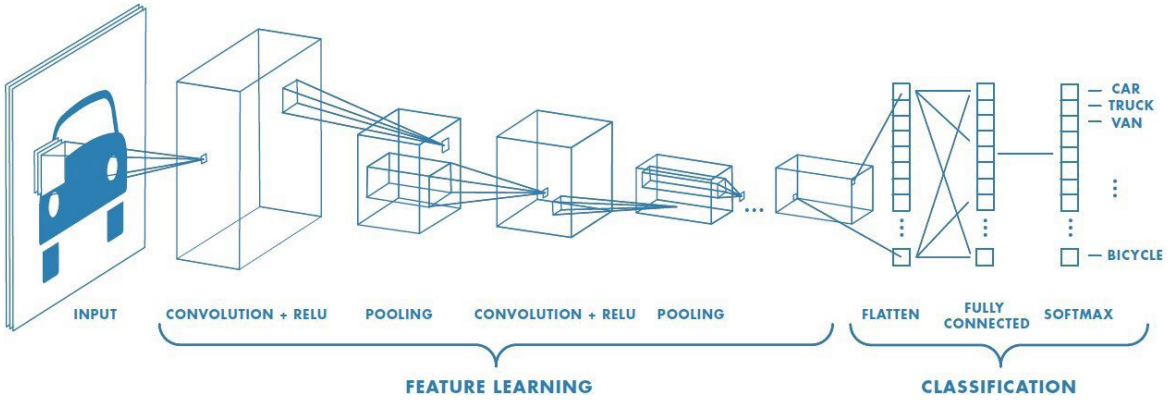


Static versus Dynamic ANNs

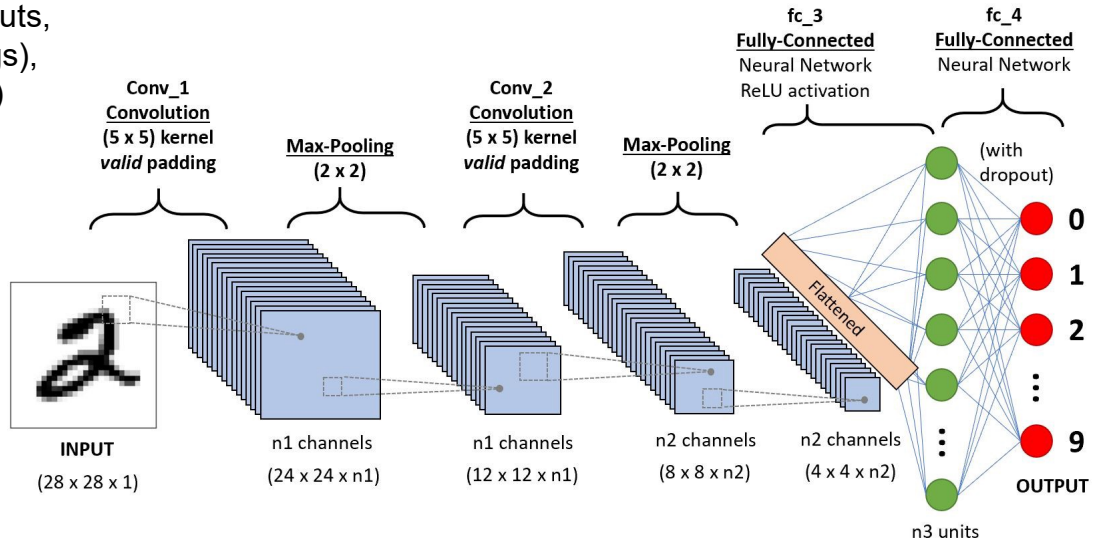
- **Static networks** – output calculated from the input through feedforward connections

- **Dynamic** – output depends also on previous inputs, outputs or states – like the recurrent network in the previous slide – takes a while to stabilise

Deep Learning Convolution NN



(Deep learning – lots of inputs, layers (doing different things), outputs. Image processing)



Artificial Neural Networks

- Introduction ✓
- Machine Learning ✓
- **History**
- Creating/Training ANNs
- Data processing
- Evaluation
- A few things to finish

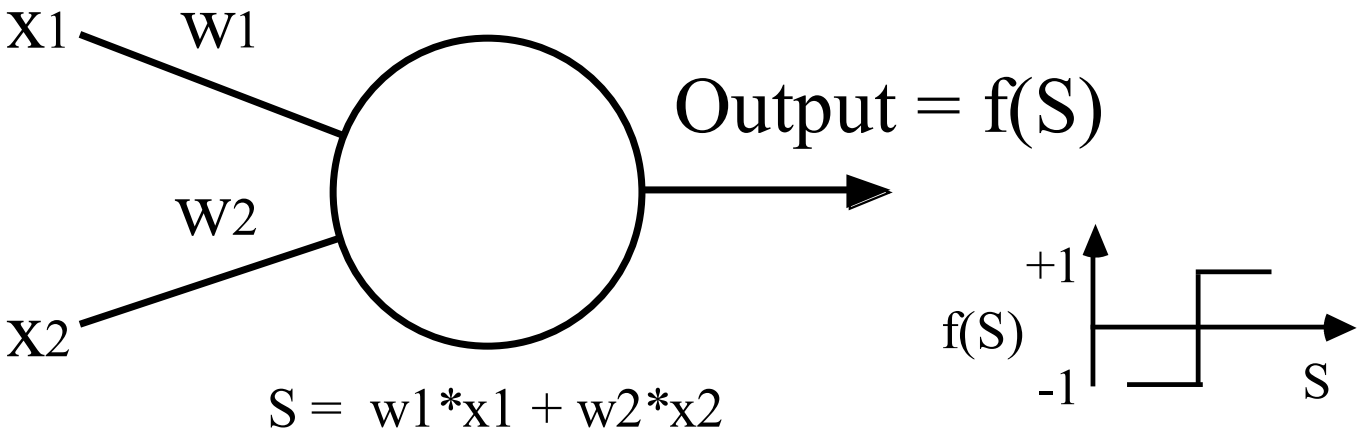
ANNs through history

- McCulloch and Pitts – 1943 (perceptron)
- Rosenblatt – 1958 (perceptron learning)
- Widrow and Hoff – 1960 (Adaline)
- Minsky and Papert – 1969 (problems)
- Kohonen – 1982 (SOM)
- Rumelhart and McClelland – 1986 (MLP)

Earliest work – 1943 – the Perceptron

- Warren McCulloch and Walter Pitts.
- Mathematical model of a single neuron that '*thresholded*' a weighted sum of inputs to produce an output.

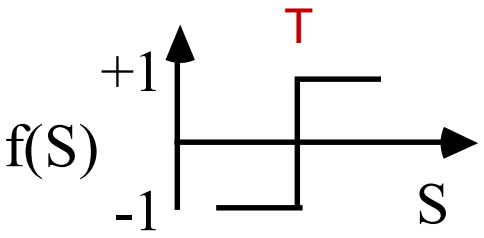
(two inputs in this example)



Earliest work – 1943 – the Perceptron

Threshold function

$S = w_1x_1 + w_2x_2$

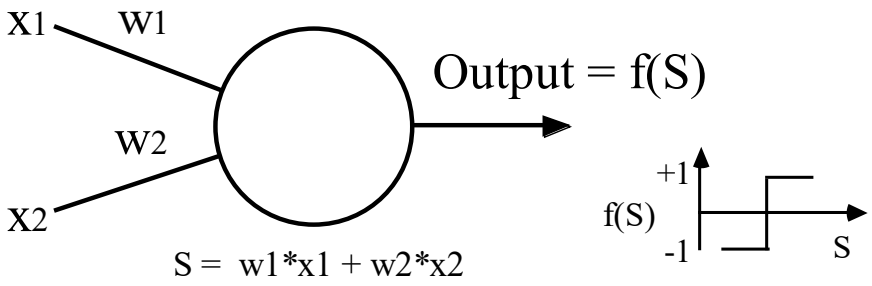


f(S)= If S>T output +1
Else output -1

Usually set T to zero (but works just as well with any other number)

The perceptron

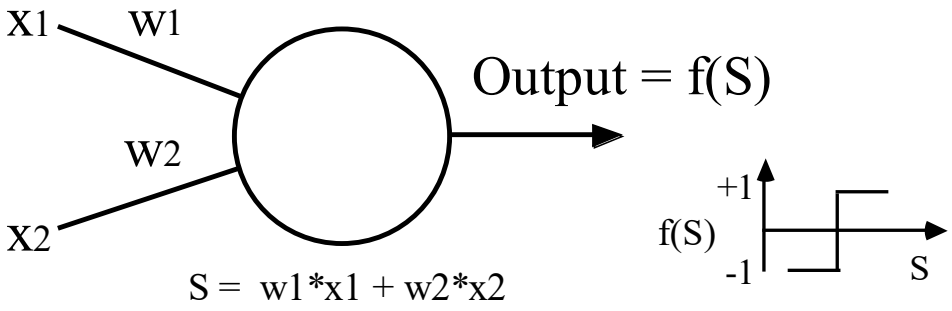
- Example so far has just two inputs
- Main drawback of McCulloch and Pitts perceptron – the weights are fixed and the model cannot learn from examples

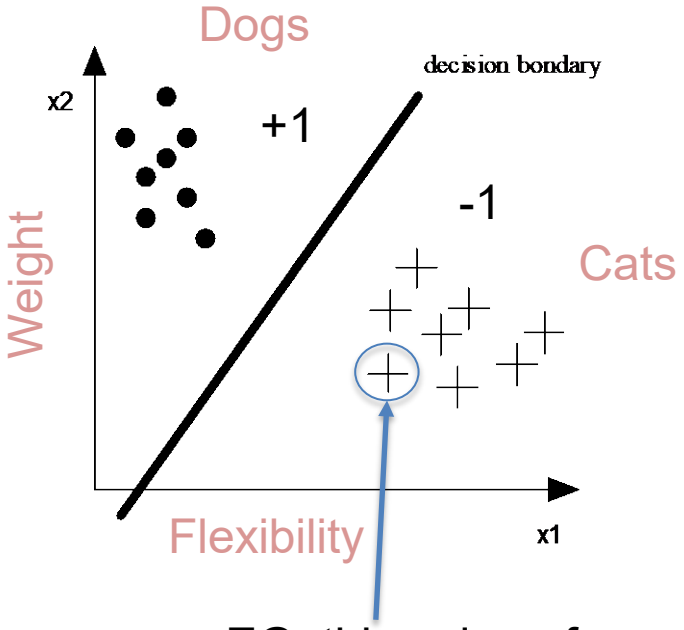


The perceptron – example with 2 inputs

decision boundary = $W.x = f(w_1, w_2)$

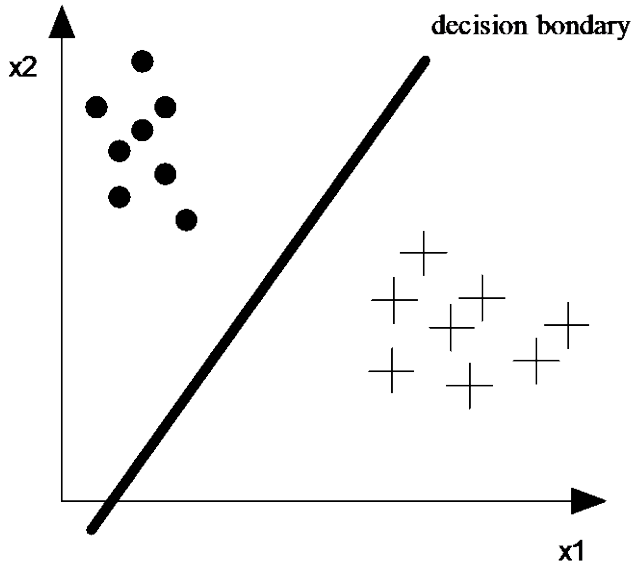
$S = W.x = W_1x_1 + W_2x_2$

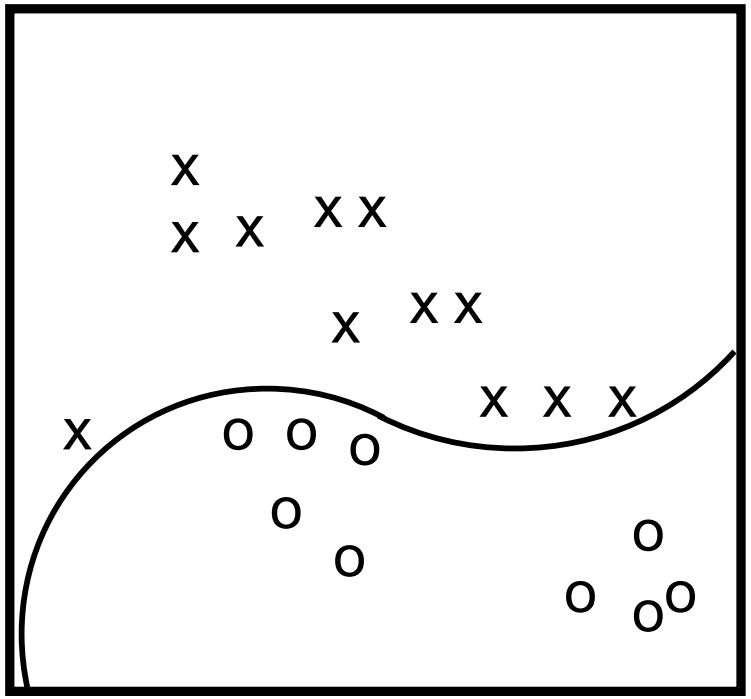




EG, this value of x_1, x_2 gives a value of S below the threshold so f(S) = -1

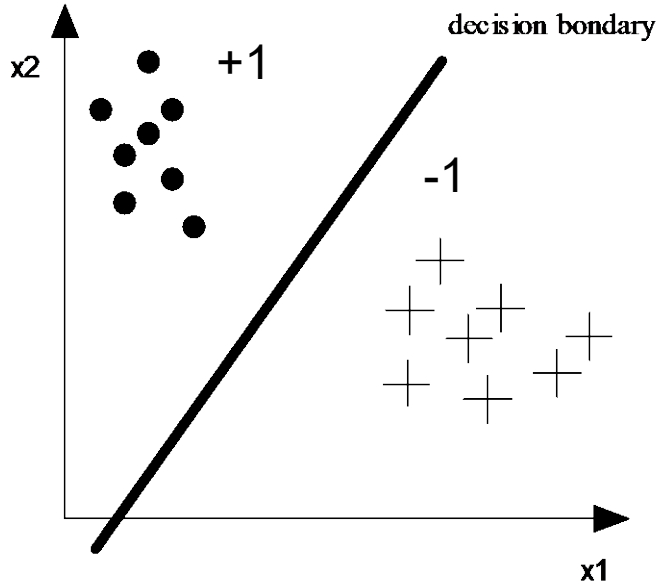
Separating Data





- At the moment we are separating data by a straight line
- If we want a curved line – we need more complex networks
- Not just a single perceptron – more on this later

The perceptron – example with 2 inputs



Some sample data

x_1	x_2	Class
1	4	+1
2	9	+1
5	6	+1
4	5	+1
6	0.7	-1
1	1.5	-1

- Note – this is not your typical y = f(x) graph
- x_1 and x_2 are independent variables (our two inputs)
- We are simply using this graph to show the category (+1 or -1) where samples with particular (x_1, x_2) values fall

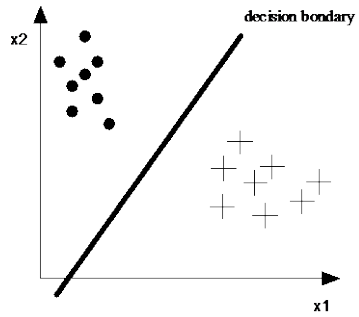
1958 – Perceptron Learning

- In the 1950s – we found out a way to ‘train’ the perceptron
- Frank Rosenblatt - proposed the **perceptron model**
- Has weights that are now **adjustable** and can be adjusted by the **perceptron learning law**

1958 – Perceptron Learning

(straight line)

- Shown to converge for **<u>linearly</u> separable** pattern classification problems



- **Multi-layer** perceptron was shown to be able to classify non-linearly separable problems

- BUT – no way to train multi-layers yet.

1958 – Perceptron Learning

- Rosenblatt's perceptron convergence theorem:

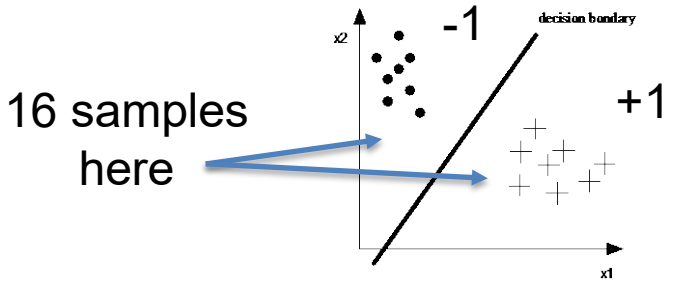
- ***If there is a set of weights that correctly classify the (linearly separable) training patterns, then the learning algorithm will find one such weight set in a finite number of iterations***

$$S = W.x = W_1x_1 + W_2x_2$$

Set of weights – in this case W_1 and W_2

1958 – Perceptron Learning

Assumptions:



- **At least one such set of weights, w^*, exists, and**
- **There are a finite number of training patterns.**
- **The threshold function results in +1 or -1.**

Perceptron Learning

- Perceptron Learning – see separate slides

The decline - 1969

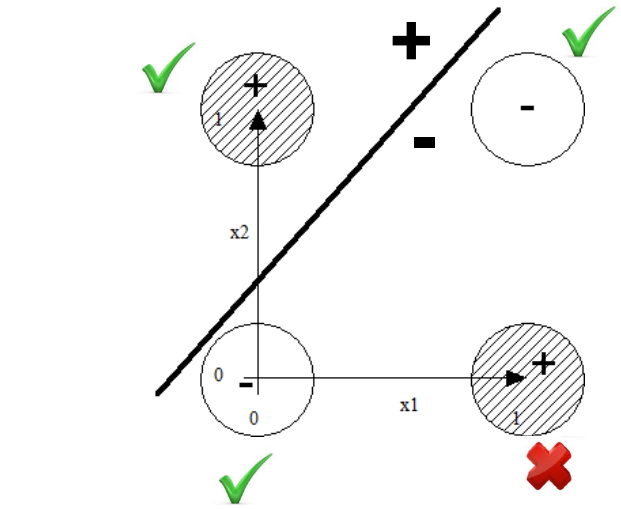
- Minsky and Papert pointed out significant problems with the perceptron

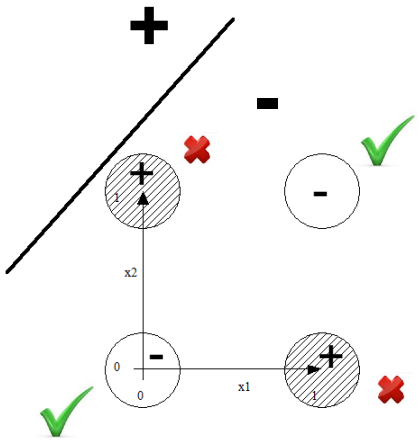
- - it cannot not solve **linearly inseparable** problems such as the simple **XOR**.

Perceptron Training

- Applied to the XOR problem - could train our perceptron to classify three of the four training examples correctly but not four.

- Problem - having correctly classified 3/4 examples, at the next step it might select the training example that is misclassified, adjust weights and end up classifying only 2/4 – ie worse.

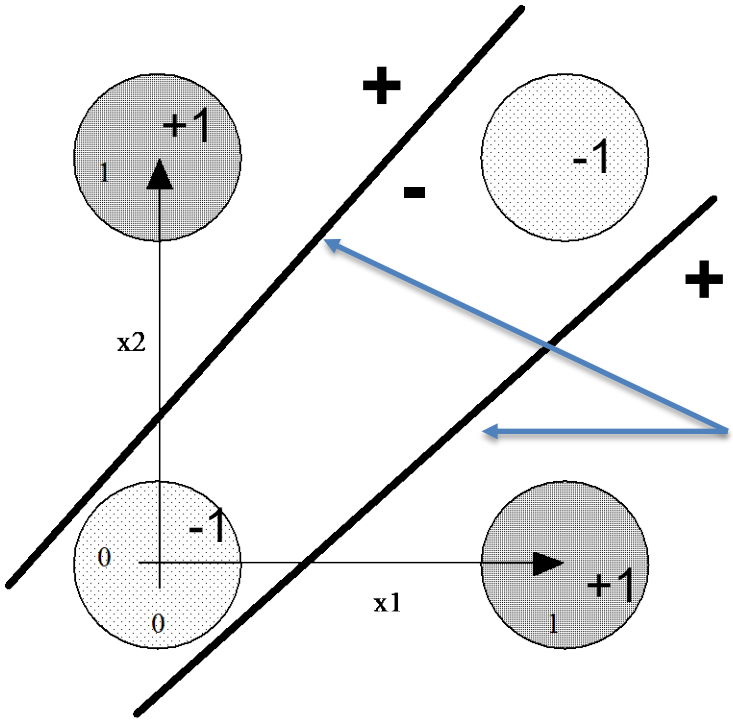




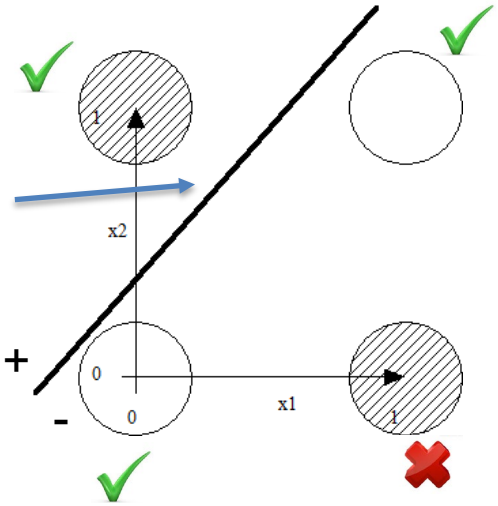
x1	x2	C
0	0	-1
0	1	+1
1	0	+1
1	1	-1

XOR Problem

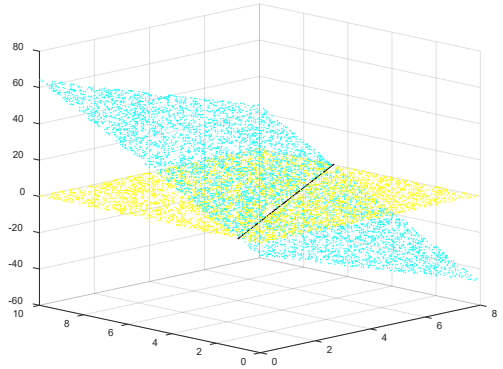
Need two lines to separate XOR - this comes from thinking in 3D. We are looking down on the surface now.



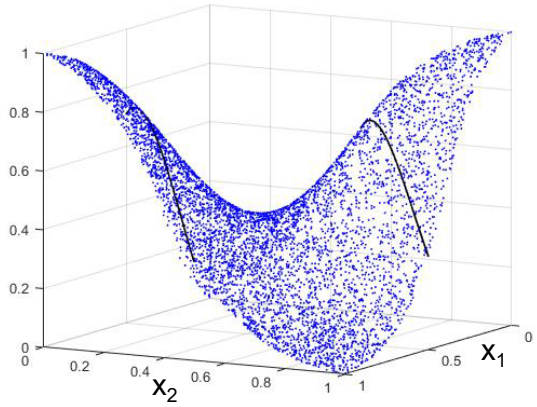
Perceptron – can only produce a single straight line as the plane passes through Wx=0

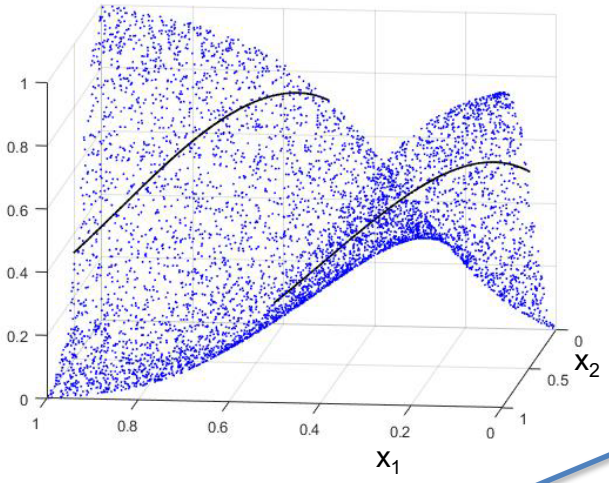


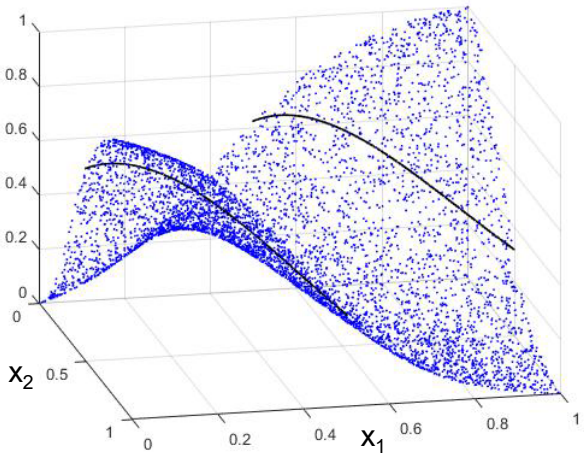
These two lines are part of a surface rising and falling. The lines represent the height at say 0.5

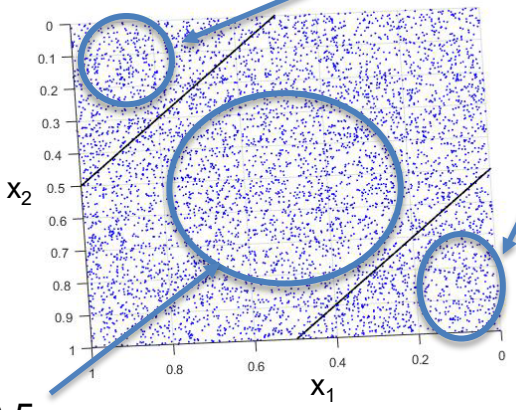


XOR with an MLP



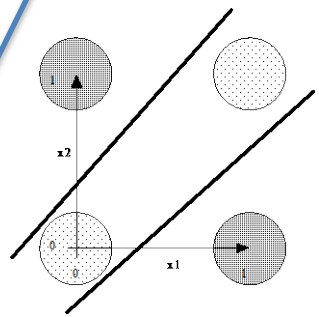






(I am now modelling an output of 0 or 1; not +1 -1)

Values here >0.5



Looking from above

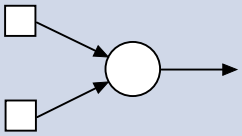
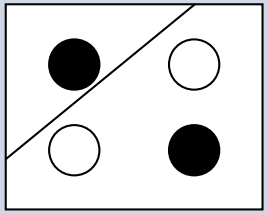
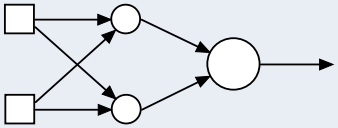
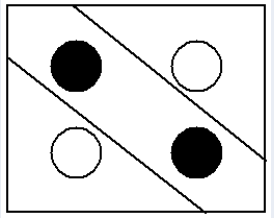
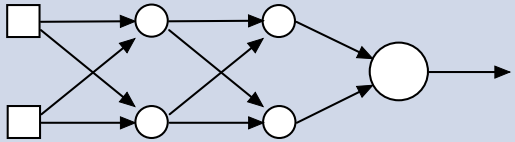
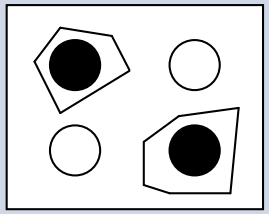
Values here <0.5

Loughborough University

The decline - 1969

- The solution, which was recognised for a long time, was to use multiple perceptrons in layers
- However, no training algorithm was available that could train these structures and research in ANNs declined.

The decline - 1969

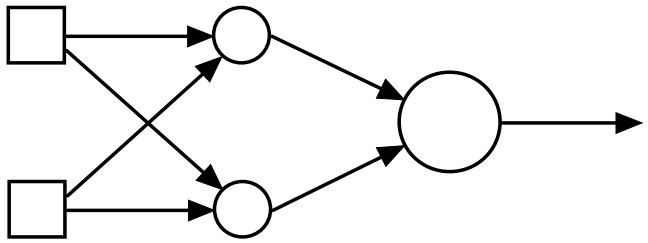
Structure	Layers	XOR Problem
	Single	
	Two	
	Three	

Although we can achieve this with Two – using different transfer functions

The renaissance – 1986+

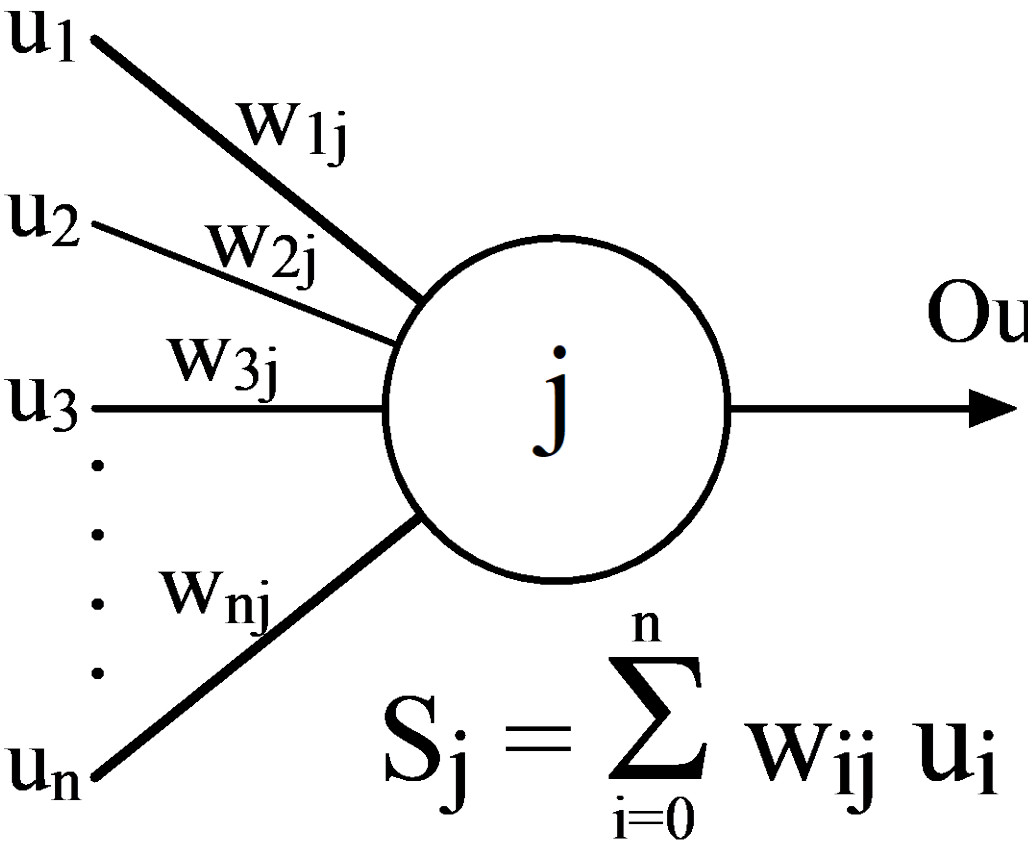
- Rumelhart and McClelland developed the MLP – **multi-layer perceptron** - and introduced a means by which it could be trained - backpropagation.

- This led to a revival in ANN research.



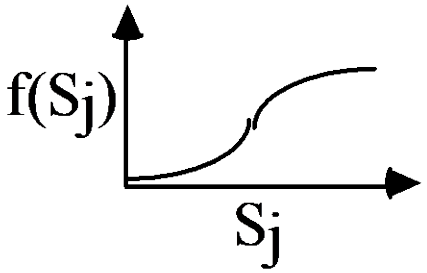
The Perceptron – activation functions

To train an MLP we need a different activation function:



$$f(S) = \frac{1}{1 + e^{-S}}$$

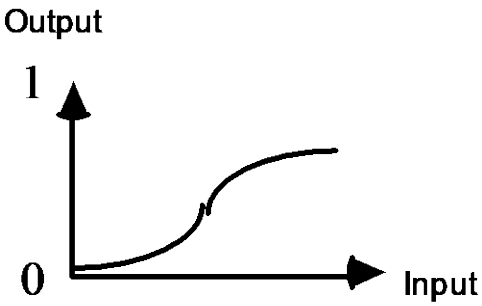
$$\text{Output} = f(S_j)$$



$$S_j = \sum_{i=0}^{n} w_{ij}\, u_i$$

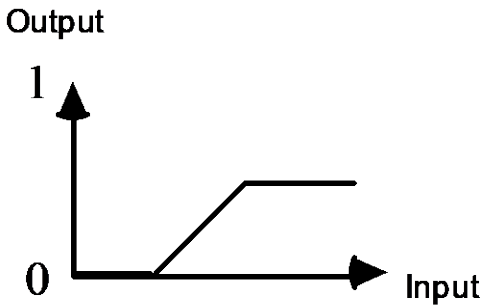
(Note – lots of perceptrons now so represent each one uniquely with j)

Activation Functions

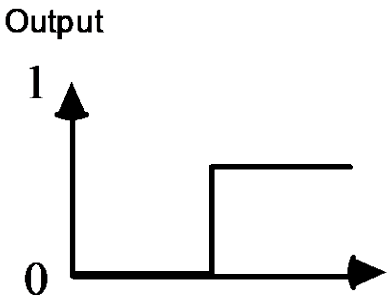


Sigmoidal

$$f(S) = \frac{1}{1 + e^{-S}}$$



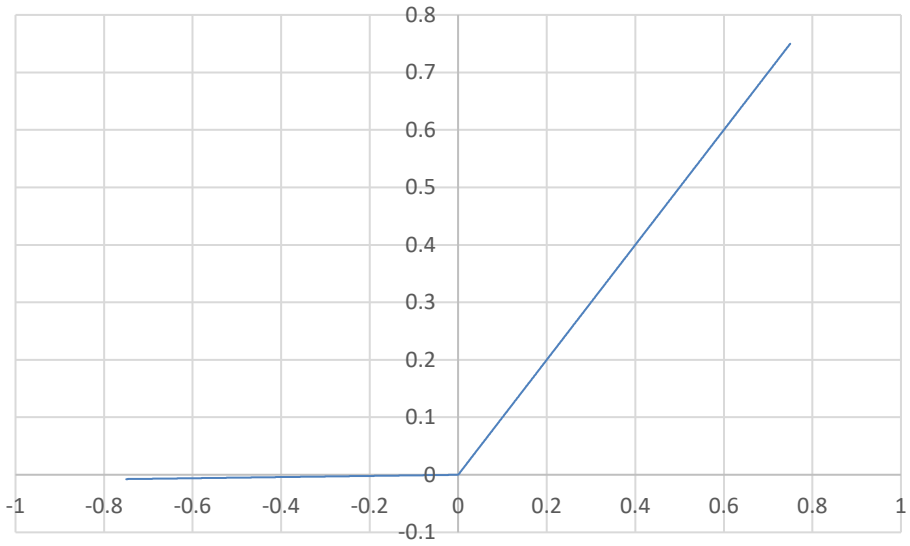
Linear threshold



Step / threshold

Rectifier

- Rectified Linear Unit (ReLU)
- Popular from 2018 in deep learning ANNs



$$f(x) = \begin{cases} x & \text{If } x>0 \\ 0.01x & \text{Otherwise} \end{cases}$$

Leaky ReLU

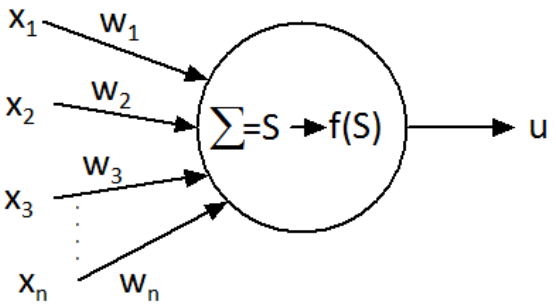
Activation Functions - a few more

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons		compet

[From Hagan]

Single cell - exercise

- Calculate the cell's output response (u)



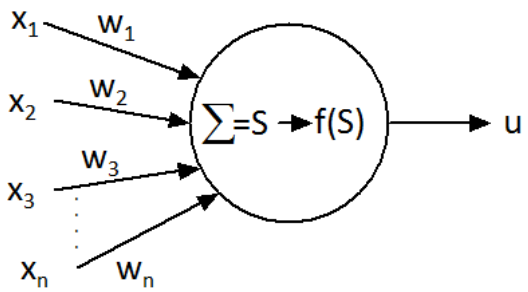
Input	Input value	Weight
1	2.7	0.23
2	3.2	-0.56
3	0.56	0.14
4	2.7	0.83

$$f(S) = \frac{1}{1 + e^{-S}}$$

Single cell exercise

- Calculate the cell's output response (u)

Input	Input value	Weight	WxInput
1	2.7	0.23	0.621
2	3.2	-0.56	-1.792
3	0.56	0.14	0.0784
4	2.7	0.83	2.241
		Σ=	1.1484

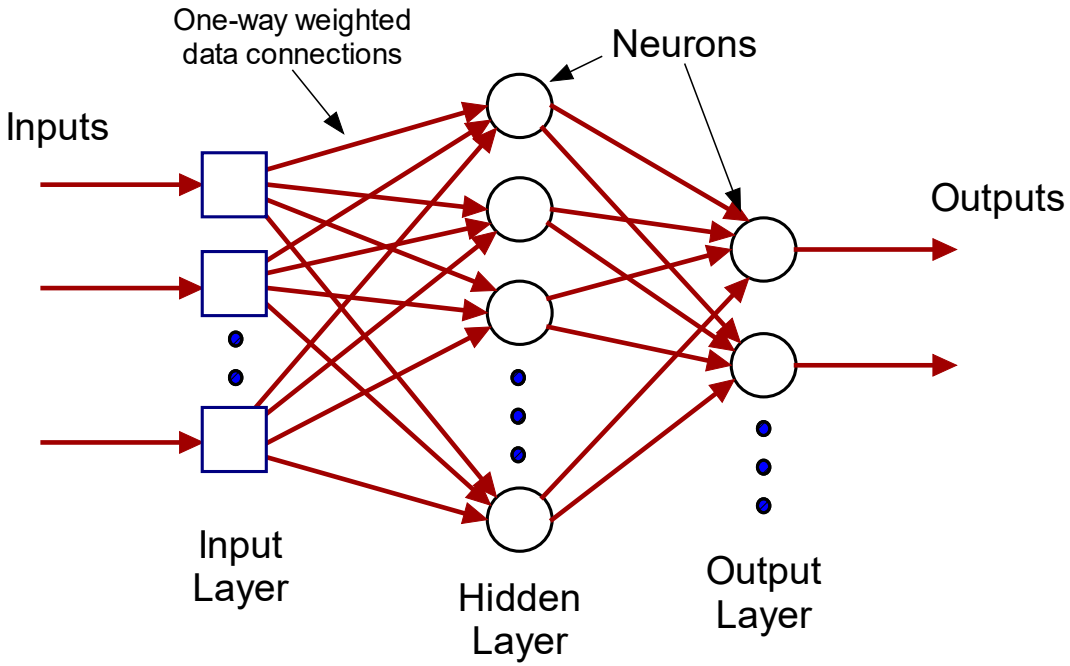


$$f(S) = \frac{1}{1 + e^{-S}}$$

$$u = f(S) = \frac{1}{1 + e^{-1.1484}} = 0.7592$$

MLP Training

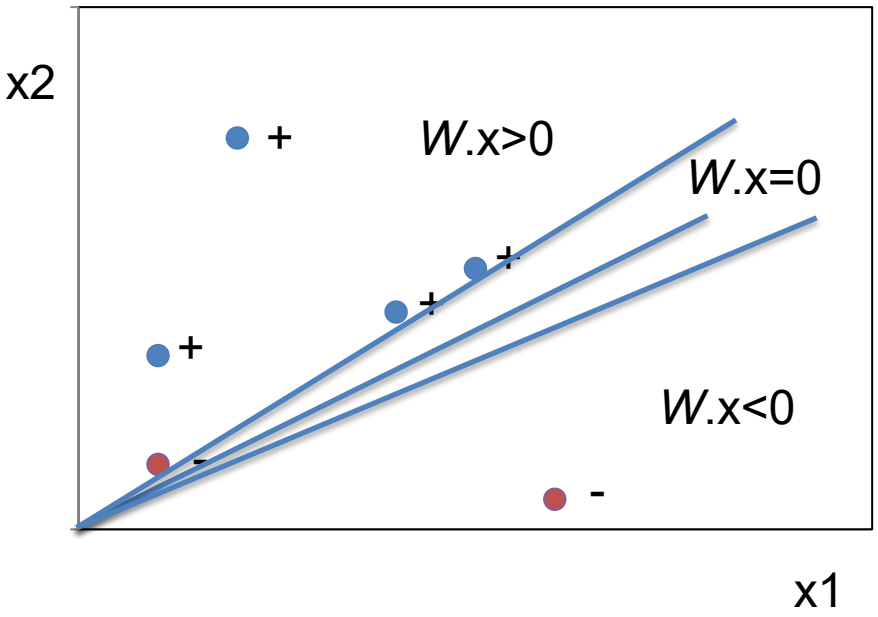
- We will look at training the MLP later – lots to discuss before we can do that

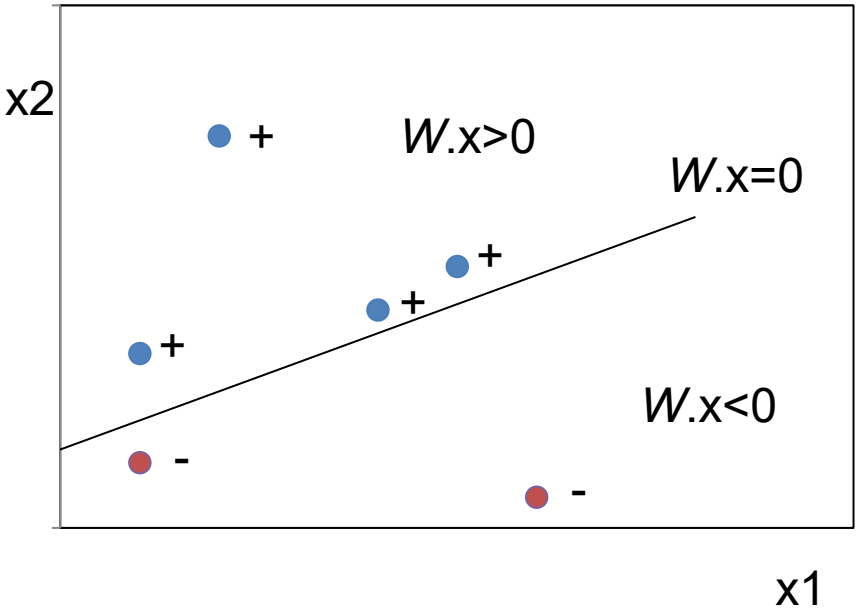


Interlude - Biases

- Biases are added to the ANN structure by convention (just like the perceptron)
- They enable the network to model more complex relationships
- Each node has an additional input equal to unity (1) with its own weight (the bias) which is adjusted during training.
- (remember the perceptron) -

Interlude - Biases





Artificial Neural Networks

- Introduction ✓
- Machine Learning ✓
- History ✓
- **Creating/Training ANNs**
- Data processing
- Evaluation
- A few things to finish

Creating and training ANNs

- In a nutshell:
  - Process the data
  - Train the network(s)
  - Assess the results

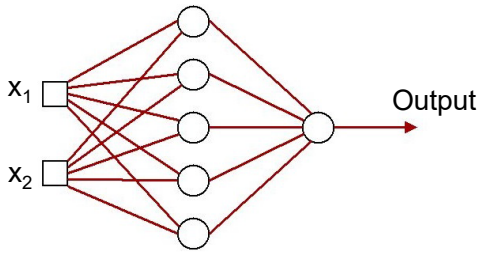
Coursework - overview

- Data pre-processing (cleansing and data splitting) – 15%;
- Implementation of the MLP algorithm (including modifications / improvements) – 35%
- Training and network selection – 20%;
- Evaluation of final model (including comparisons between different modifications to the algorithm) – 20%;
- Comparison with another data driven model – 10%;

ANN modelling sequence

STEP 1. **Gather data** - Ensure sufficient data are available for a meaningful study in terms of both quantity and quality

STEP 2. Select predictand(s) (what are we trying to predict?)



STEP 3. **Data preprocessing (Stage 1)**

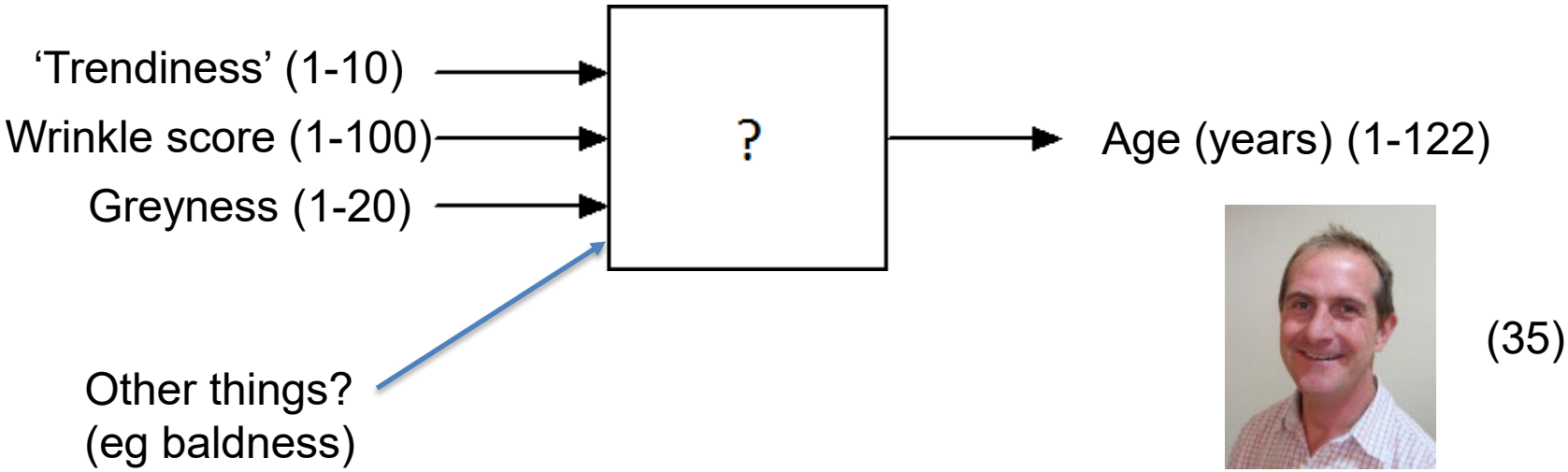
3.1 Data cleansing - Remove significant underlying upward or downward trends (eg, using first differences or remove regression line), seasonal components, missing data, errors.

3.2 Predictors - Identify the most significant predictors for the chosen predictand. If necessary, reduce the number of predictors by means of principal components. Determine suitable lag times for each predictor and calculate appropriate moving averages for the predictors when applicable.

Machine Learning – example difficult task

Estimating a person's age

	Predictors			Predictand
	Trendiness	Wrinkles	Greyness	Age
Person 1	5	45	2	22
Person 2	9	23	1	19
Person 3	3	72	18	38
etc	etc	etc	etc	etc



ANN modelling sequence

STEP 4. ANN Selection

4.1 Network type - Select the most appropriate network type for the application.

4.2 Training algorithm - Select a suitable training algorithm to modify weights and biases and determine network architecture. Choose appropriate values for learning parameters (momentum and learning rate).

STEP 5. Data preprocessing (Stage 2)

5.1 Data standardization - According to the algorithm chosen, standardize data to the ranges [0,1], [-1,1], [0.1,0.9], etc., or normalize the data.

5.2 Data sets - Create cross validation data sets by splitting the data into appropriate calibration, testing and validation sets. With a large data set this is relatively easy as the data can be split into three representative sets. However, for smaller data sets, cross training should be used

ANN modelling sequence

STEP 6. Network Training

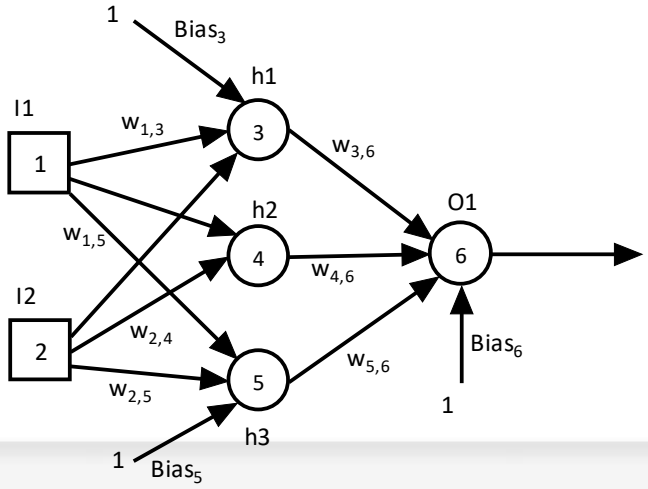
6.1 Architecture - Specify the number of hidden layers and number of nodes in these layers. This may be unnecessary if a pruning algorithm, or cascade correlation is used. Begin with one hidden layer.

6.2 Training - Train a number of networks using the calibration and validation data. Terminate the training process when results from the validation data indicate over-fitting to the calibration set.

STEP 7. Evaluation - Select error measures that are appropriate to the model output and purpose. Compare results with those derived from alternative model configurations and alternative methods.

Representation of the ANN

- From an holistic viewpoint an ANN is simply a collection of numbers.
- These numbers represent the weights connecting nodes
- And the biases for each of those nodes.



(Bias$_4$ not shown)

Representation of the ANN

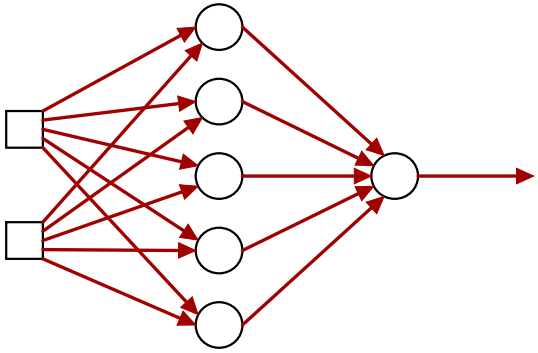
- If we know the ANN structure (number of inputs, hidden, output nodes) and the weights and biases we can ‘recreate’ that ANN.

- Thus we can ‘store’ an ANN as a collection of these numbers.

- How we structure these numbers is up to us – eg an array, a list, an object, etc.

Representation of the ANN - exercise

- Consider how you might represent an ANN in a program
- How are you going to structure and store the data?
- If you want to recreate this ANN how will you 'store' it (in a text file say, or as an object, etc.)?
- For example -
- 2D array for weights inputs to hidden nodes;
- 1D array for weights hidden nodes to 1 output node;
- 1D array for biases on hidden nodes;
- Single real variable for bias on output node.



Creating and training ANNs

- In a nutshell:
  - **Process the data**
  - Train the network(s)
  - Assess the results

Example data set

- These data are from the UK Flood Estimation Handbook
- Data for over 1,000 catchments in UK

- DTM area – drainage area in km^2
- BFIHost – underlying baseflow measure
- FARL – flood attenuation due to reservoirs
- SAAR – Standard period average annual rainfall (mm)
- Index Flood - the median of the AMS (cumecs)

Fishlake, UK, 2019

Example data set (notice different ranges)

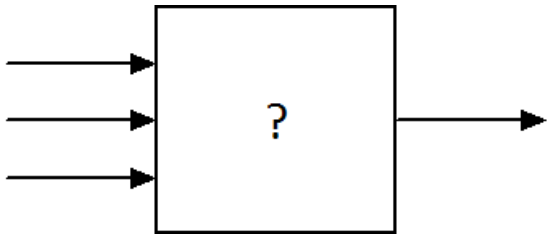
	Inputs (4)			Correct Output C
DTM AREA	**BFIHOST**	**FARL**	**SAAR**	**Index flood**
1800.92	0.471	0.959	1110	183.0255
31.4	0.514	0.998	922	9.581
130.1	0.653	0.985	596	3.815
344.36	0.461	0.909	1170	80.671
710.96	0.458	0.995	1153	200.849
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
	.	.	.	.

Predictors **Predictand**

Data processing

Predictor 1	Predictor 2	Predictor 3	Predictand
0.234	6.733	2001	27.91
1.253	4.533	2003	-999
6.789	5.675	3011	25.33
.	.	.	.
12.453	2.333	513	800

- Arranging data into a file so your ANN can read/process it
- Missing data, spurious values?
- Splitting data into subsets
- Standardisation



Missing data / spurious data

- Eg -999; temperature data>56.7ºC* etc.
- Changes from one day to next>20ºC

- Maybe explore (remove) outliers that are over 2 or 3 standard deviations from the mean

- Remove (not time series) or Interpolate (time series)

- *Imputation* – official statistical term for replacing missing data with substitute values

* Death valley, 10 July 1913

Splitting data into subsets

- Before I explain how we might split our data into subsets for training – we need to discuss a couple of issues – ***Bias*** and ***Variance*** **–** to explain why we split our data set

Under- / Over-fitting

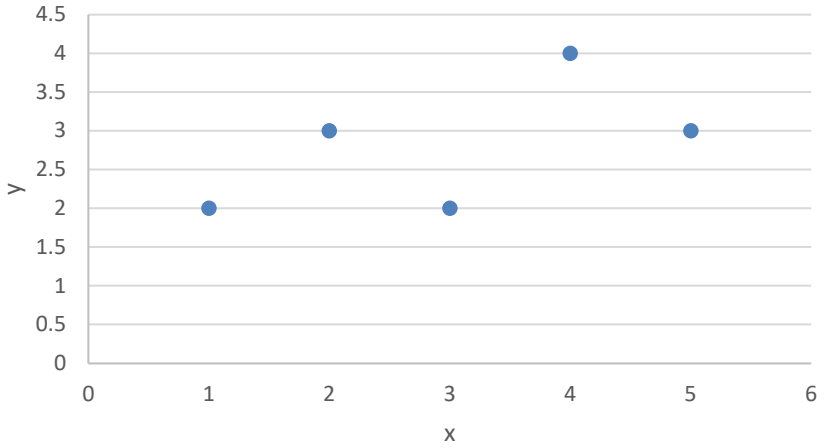
$$y = f(x)$$

x	y
1	2
2	3
3	2
4	4
5	3

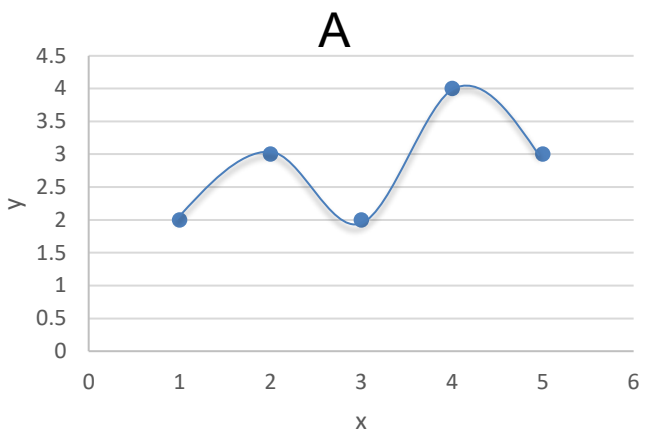
Consider this: y is some (unknown) function of x

We don't know what the function is

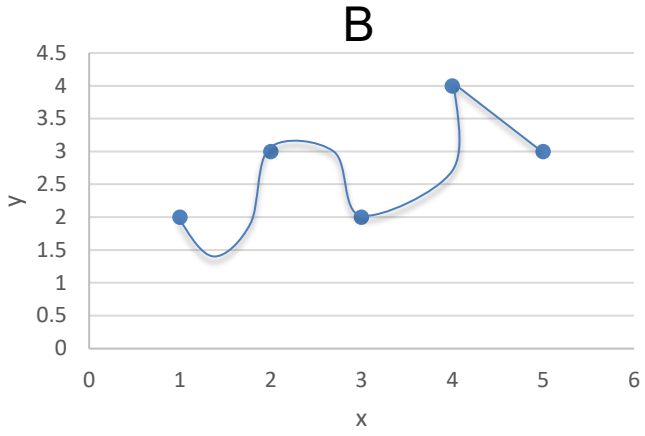
but these data are available so we can build a model to try and model the function -



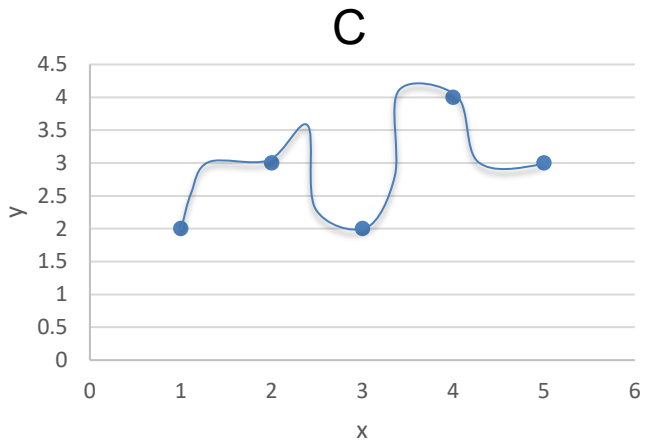
The real function might be -



or



or

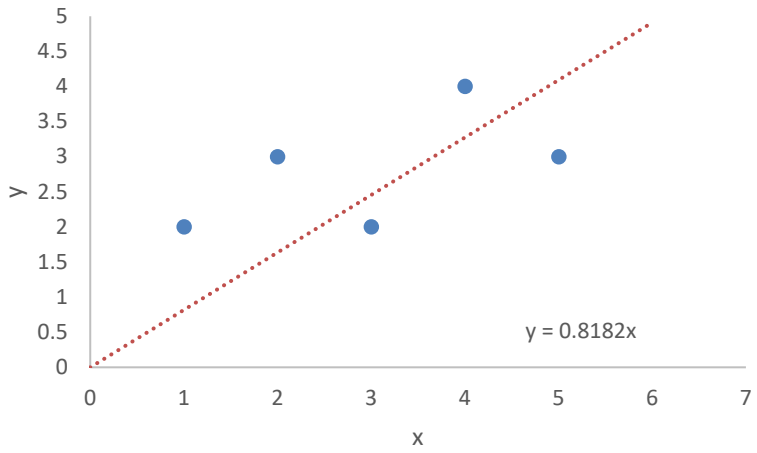


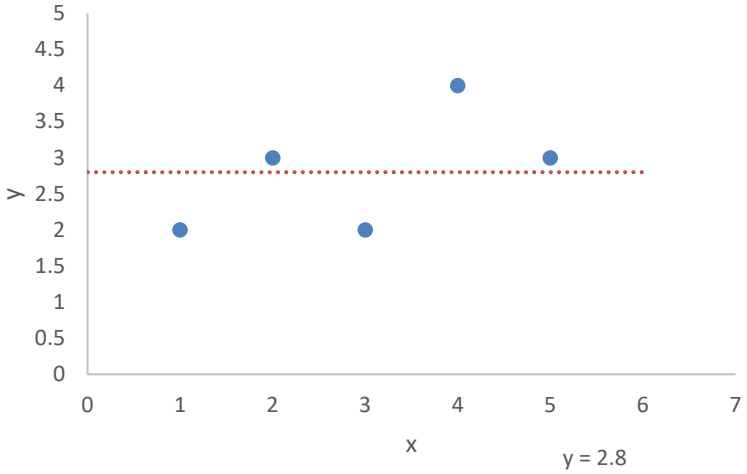
or something else entirely

We just don't know – all we have is 5 sample data points to work with

EG when x=1.5 what is y?
Which model gives the best answer?
We don't know as we don't know the value of y when x=1.5

Fitting a simple one parameter model





RMSE = 1.02

(y=mean of all values)

RMSE = 0.75

This is the best one parameter model here

RMSE=Root Mean Squared Error: smaller is better!

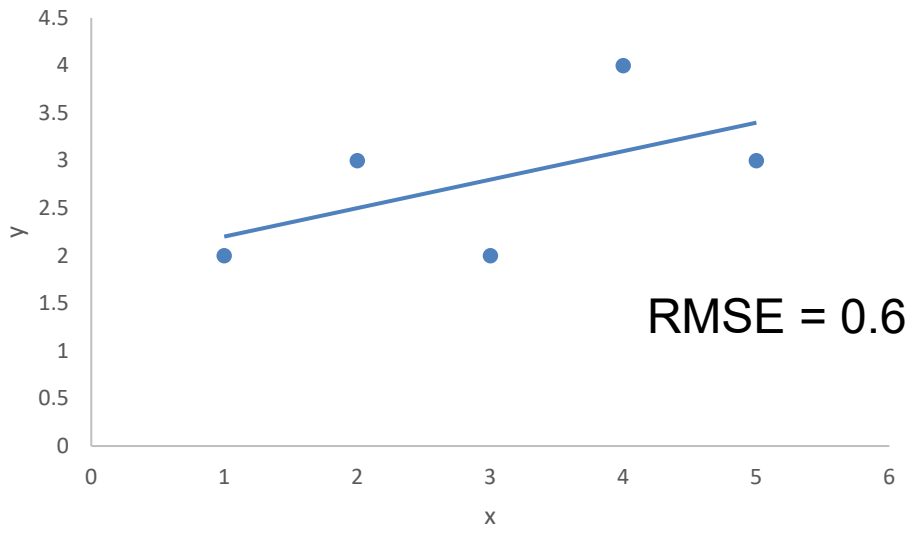
Bias and variance

We are going to try to fit a polynomial function to the data.

(We could use a machine learning technique)

With two parameters in our model (0.3 and 1.9) we can create an approximation to the function as a straight line.

$$y = 0.3x + 1.9$$

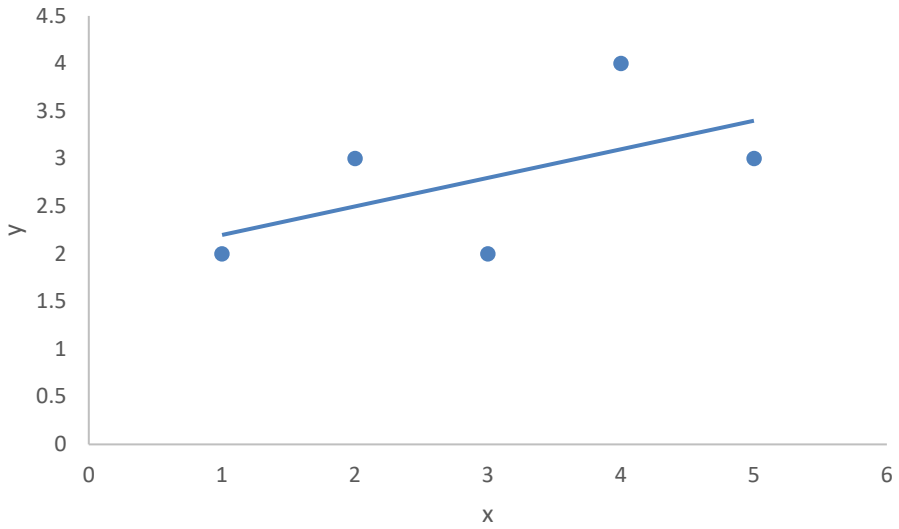


RMSE = 0.6164

[This is already better than the mean model from the last slide (RMSE=0.75)]

Bias and variance

$$y = 0.3x + 1.9$$

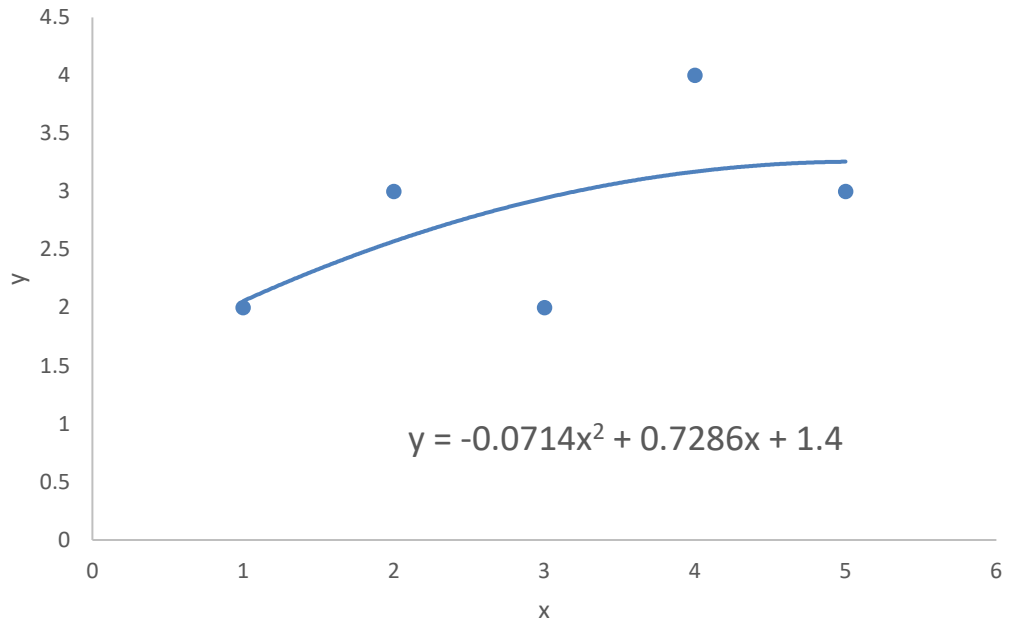


RMSE = 0.6164

- Too few parameters, can't fit the data
- Would not predict new data well
- This model has a high **bias**

Bias and variance

Adding another parameter gives a curve which might be better:

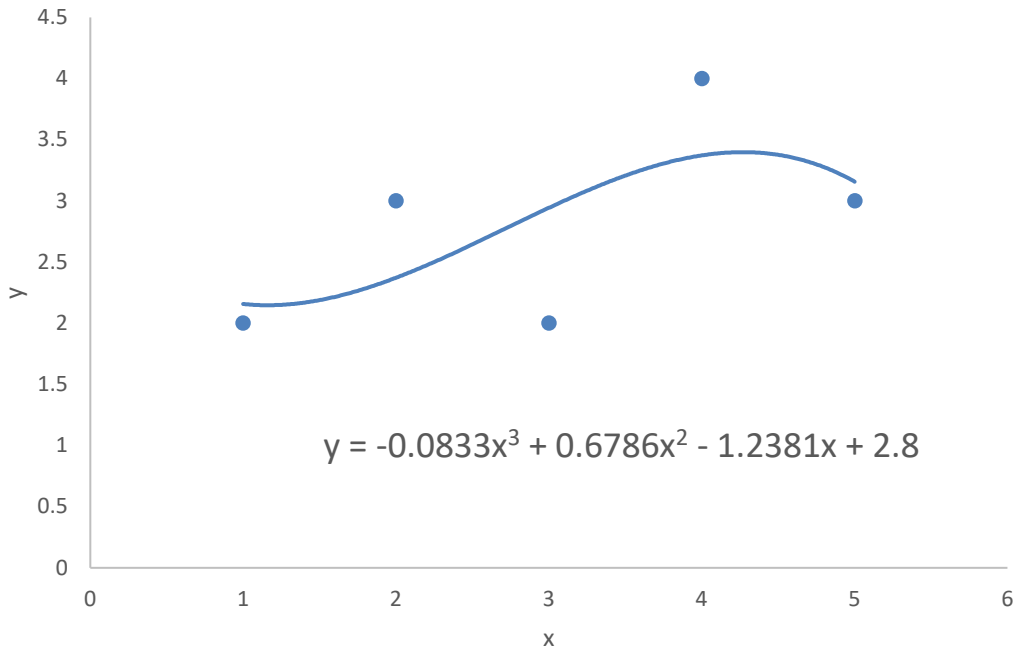


3 parameters

RMSE = 0.6050

Bias and variance

Adding another parameter:

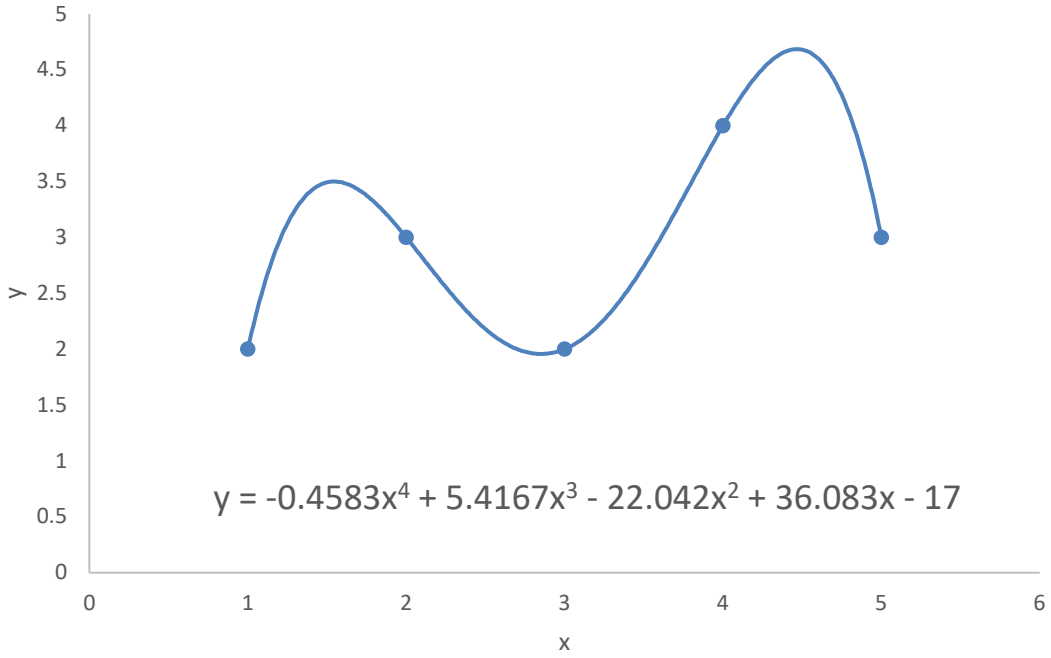


4 parameters

RMSE = 0.5880

Over fitting - variance

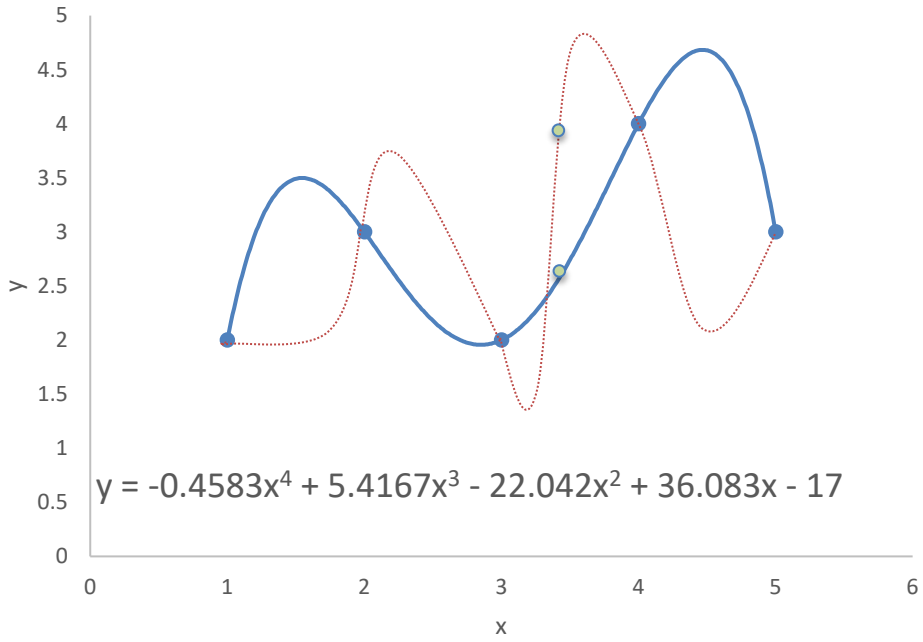
5 parameters – can model 5 data points perfectly – so why don't we?



RMSE = 0.0070

Over-fitting: also wrongfully learning meaningless patterns that are unique to the training data

Over fitting - variance



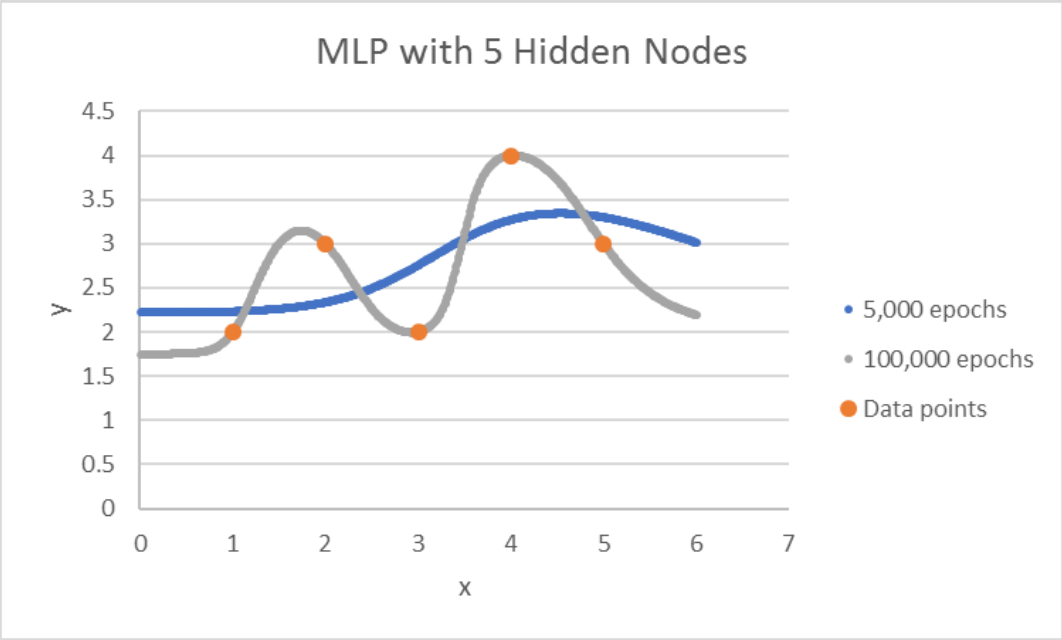
- Too many parameters, fits the data perfectly
- Would not predict new data well (same as when high bias)
- Model has a high **variance**

My 5 parameter model

Actual unknown function

Eg – my model's attempt at modelling y when x=3.5:
My model gives y~2.5; actual is y~4.

MLP training

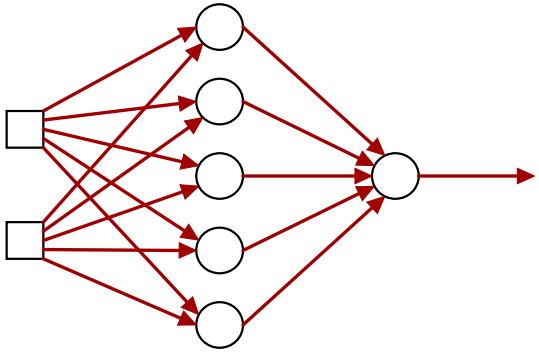


- We want something in between.
- Powerful enough to represent the underlying structure
- Not too powerful that it cannot generalise

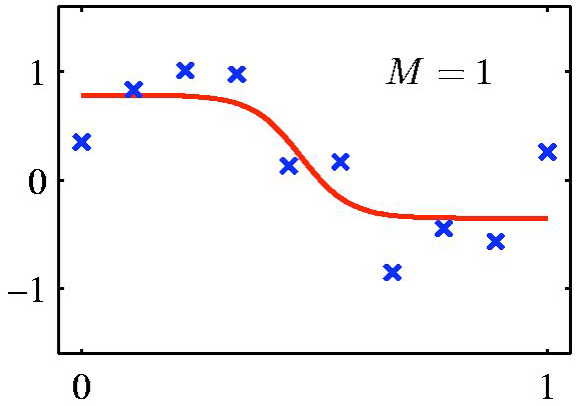
(Why use an MLP on a problem like this?)

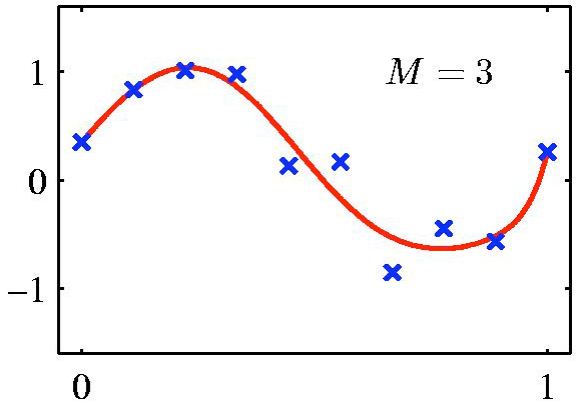
Exercise

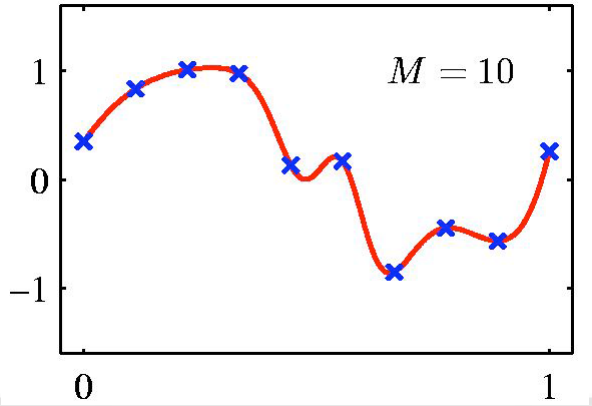
- How many parameters are in this model?
- 2-5-1 MLP

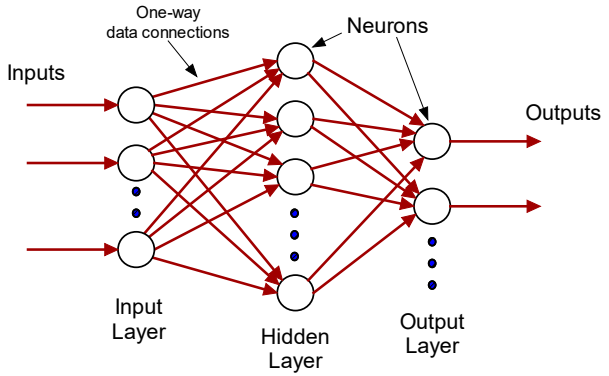


Hidden nodes ~ parameters in the model



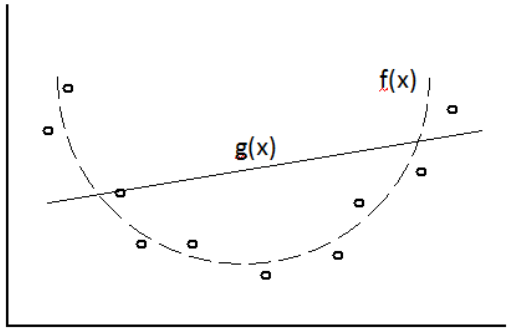


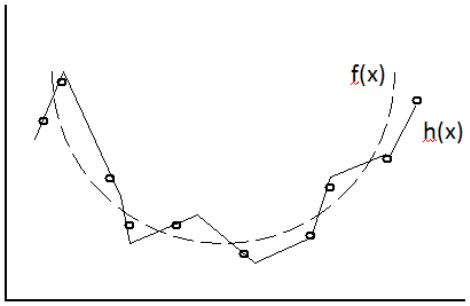


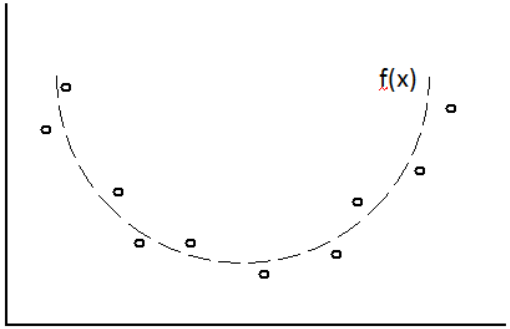


Trade-off between bias and variance

- So, during ANN training we develop a model that can model the underlying function reasonably well, but is not over-trained on the data

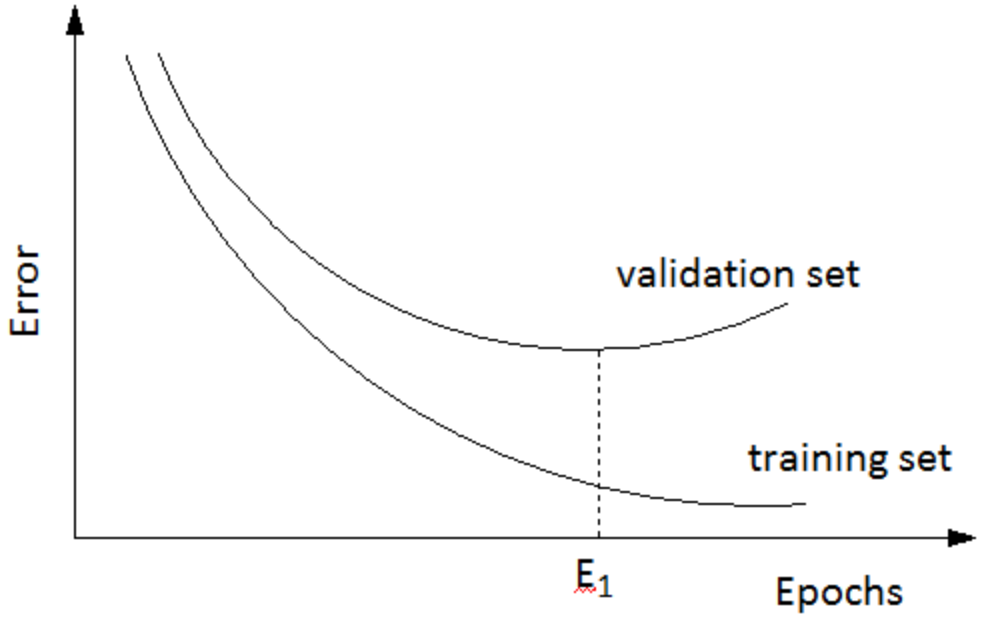






Avoiding over-fitting

- Early stopping
- Training data and unseen **validation** data



Note – this is an 'idealised' representation of the error for these two sets.
In practice we may say less smooth/rising and falling than this.
It may be worth continuing training a bit further to see if anything changes.

Splitting data into subsets

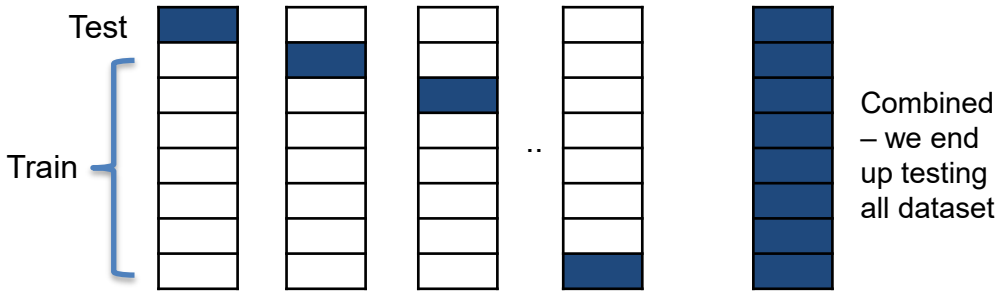
(I want my son to be good at Tennis – but can't afford to join the tennis club – but can afford to hire the odd badminton and squash courts)

- Training set (eg 60%) (Badminton)
- Validation set (eg 20%) (Squash)
- Test set (eg 20%) (Tennis)

- Random splitting (eg avoiding seasonal differences in rainfall for a single year of data)

Cross-validation – when data are sparse

- K-fold Cross-validation – when data are sparse
- The data are split into S segments (aka folds).
- A network is trained on S-1 of these segments and tested on the remaining segment.
- This is repeated S times so that each segment has a turn at being left out and tested on.
- Typical value for S is 10.



Standardisation

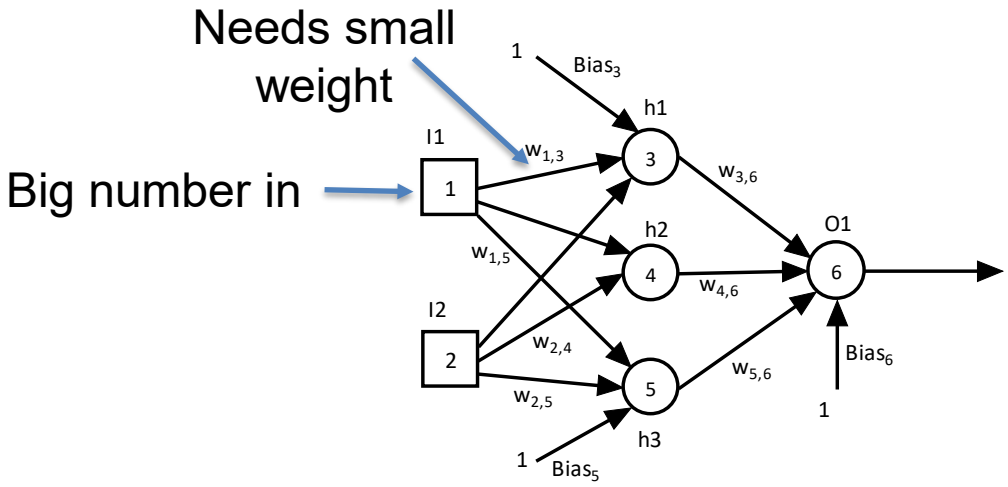
- Before we can use our data set – we should **standardise** the data

- This means reducing the range of each of the predictors (inputs) and the predictand (output) to a ‘reasonable’, consistent range.

Standardisation

- Without standardisation, large values input to an ANN would cause a number a problems:

i) Very small weights would need to be applied to avoid large input values overpowering the neurons in the hidden layer. These small weights would lead to inaccuracies with floating point calculations during training.



Standardisation

ii) Some inputs will dominate the network more than others.

Predictor 1	Predictor 2	Predictor 3	Predictand
0.234	6.733	2001	27.91
1.253	4.533	2003	-999
6.789	5.675	3011	25.33
.	.	.	.
12.453	2.333	513	800

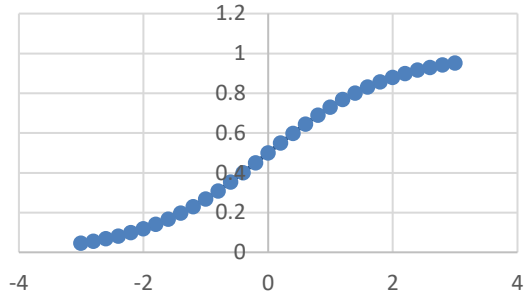
E.g. Predictor 3 would have far more influence than predictors 1 and 2 in this case

Standardisation

iii) With large input values, unless we use extremely small initial weights, changes made by the backpropagation algorithm would be insignificantly small.

- training would be very sluggish, as the gradient of the sigmoid function at extreme values would be approximately zero.

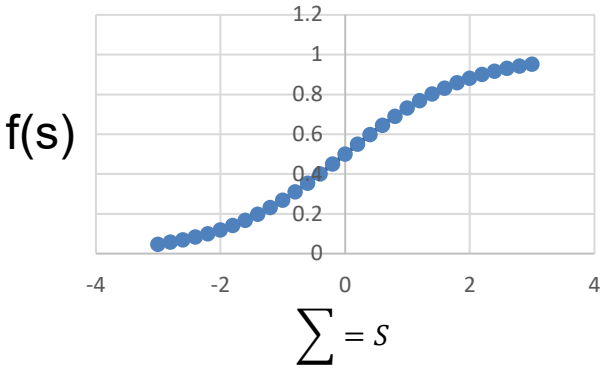
- It is this gradient that is used in the adjustment of weights and biases in an ANN during training.

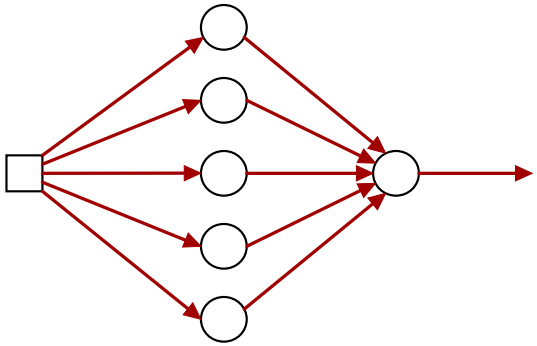


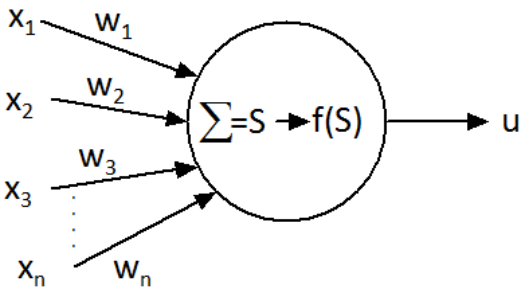
E.G values greater than 3 here – you can see the gradient of the sigmoid function is pretty much flat

Standardisation

- For the output node(s):







$$f(S) = \frac{1}{1+e^{-S}}$$

- If we have a sigmoid transfer function for the output node – our ANN can only produce real values between 0 and 1

- So – there is no point in trying to train our MLP to predict values outside this range – as it just cannot do it!

Standardisation

- However, if we standardise our training data from its current range to [0,1] – our ANN will then be able to predict these values.

- We can then de-standardise the values our ANN produces to give us the actual values we require.

Ways to standardise data (but no rules)

- With respect to the range of all values [0,1]:

$$S_i = \frac{R_i - Min}{Max - Min}$$

S_i : Standardised value

R_i : Raw value

Predictor 1 R_i	S_i
0.234	0.053
1.253	0.134
6.789	0.572
.	.
12.453	0.997

Max & Min needed

The output

- Take this small data set

Input 1	Input 2	Output
12	13	2
5	11	8
16	110	16
7	5	5

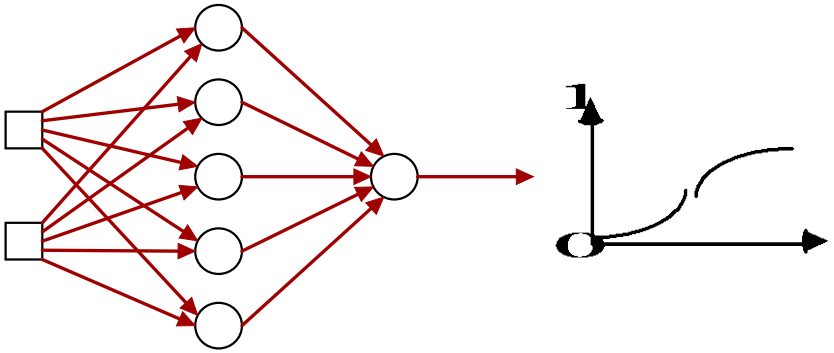
- We might standardise these data to [0,1]
- Once we have trained our MLP on these data

- it can model values between 2 and 16.

Eg to model 16 it would have learnt to output 1 (from the output node's sigmoid function) which, when de-standardised, gives 16.

- But – how can it model values outside this range in future?

The output



- The output from the MLP will be 0 – 1 (sigmoid function) which, when de-standardised, ranges from 2 to 16. How can it model, say, 22?
- Answer – by standardising the output to [0.1, 0.9]
- So, if the MLP needs to predict, say, 22; it can output, say, 0.95; which when de-standardised might give us 22.

Ways to standardise data (but no rules)

- [0,1]:

$$S_i = \frac{R_i - Min}{Max - Min}$$

S_i : Standardised value

R_i : Raw value

- [0.1,0.9]:

$$S_i = 0.8\left(\frac{R_i - Min}{Max - Min}\right) + 0.1$$

Ways to standardise data (but no rules)

- With respect to the sum of squares of all values.

$$S_i = \frac{R_i}{\sqrt{SS_i}} \qquad SS_i : \text{Sum of squares of raw values}$$

Ways to standardise data (but no rules)

- With respect to the mean and standard deviation of the data (Normalisation)

$$S_i = \frac{R_i - \mu}{\sigma}$$

μ : Mean of raw data

σ : Standard deviation of raw data

Standardisation

- Note – you must standardise all the inputs AND the output(s)
- When evaluating your MLP you must de-standardise the output to get a meaningful answer:
- Exercise - rearrange this to de-standardise the output (ie what is R_i?):

$$S_i = 0.8\left(\frac{R_i - Min}{Max - Min}\right) + 0.1$$

Standardisation

$$S_i = 0.8\left(\frac{R_i - Min}{Max - Min}\right) + 0.1$$

$$R_i = \left(\frac{S_i - 0.1}{0.8}\right)(Max - Min) + Min$$

Up to you how you do this – in your program or externally in a spreadsheet beforehand and afterwards for example

Example

Raw Values	[0,1]	[0.1,0.9]	$S_i = \frac{R_i}{\sqrt{SS_i}}$	$S_i = \frac{R_i - \mu}{\sigma}$
12	0.750	0.700	0.4636	0.947
5	0.167	0.233	0.1932	-0.821
7	0.333	0.367	0.2704	-0.316
15	1.000	0.900	0.5795	1.704
4	0.083	0.167	0.1545	-1.073
3	0.000	0.100	0.1159	-1.326
11	0.667	0.633	0.4250	0.694
9	0.500	0.500	0.3477	0.189

Standardisation - summary

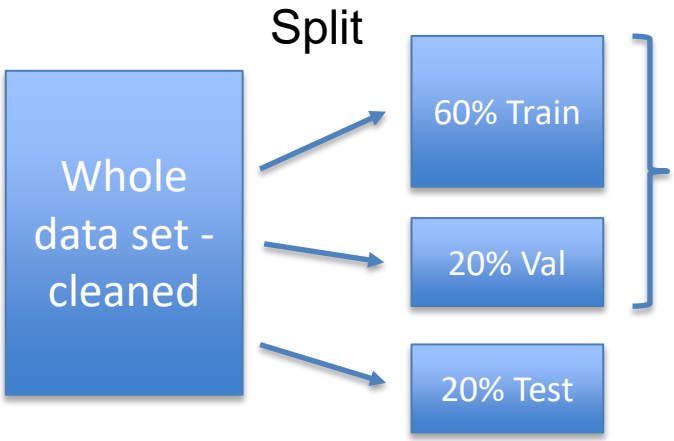
- We need to standardise the predictand to [0.1,0.9]
- We also standardise our predictors to a small range
- For consistency and simplicity – it is common to standardise the predictors and predictand(s) to the same range: [0.1, 0.9]

Standardisation - summary

$$S_i = 0.8\left(\frac{R_i - Min}{Max - Min}\right) + 0.1$$

- Strictly speaking, when we standardise our data, the Max and Min values for each predictor and the predictand should come from the Train and Validation data sets – not the Test set which is used for final testing

Standardisation- summary



Get max and min from these two for each input and output

Apply standardisation to all three data sets for each input and output using max and mins obtained

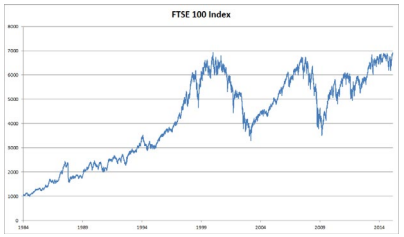
$$S_i = 0.8\left(\frac{R_i - Min}{Max - Min}\right) + 0.1$$

Another example data set – exploring lags

	FTSE 100					Gold
Date	Open	High	Low	Close	Volume	Close
12/12/2020	7129.1	7168.6	7121.7	7133.1	526735100	1567.4
13/12/2020	7133.1	7205.1	7133.1	7190.8	561646800	1575.1
14/12/2020	7190.8	7232.8	7183	7197	519107200	1582.7
15/12/2020	7197	7261.6	7185.2	7236.7	688422300	1585.8
18/12/2020	7236.7	7242.1	7209.7	7219.5	440062500	1600
19/12/2020	7219.5	7219.5	7162.8	7179.2	624124300	1607.5
.	.	.	.	.	.	.
.	.	.	.	.	.	.

- Say I wanted to predict the closing price of the FTSE 100 at least one day ahead
- These are the daily data I have managed to get to build my model

Another example data set



- How would you rearrange these data so you had appropriate predictors and the predictand?

- We want to build a one-step-ahead model to predict the FTSE 100 closing price (ie the closing price tomorrow)

Financial Times Stock Exchange 100

Another example data set

First, I would group my predictors together

	Predictors					Predictand
Date	FTSE Open	FTSE High	FTSE Low	FTSE Volume	Gold Close	FTSE Close
12/12/2020	7129.1	7168.6	7121.7	526735100	1567.4	7133.1
13/12/2020	7133.1	7205.1	7133.1	561646800	1575.1	7190.8
14/12/2020	7190.8	7232.8	7183	519107200	1582.7	7197
15/12/2020	7197	7261.6	7185.2	688422300	1585.8	7236.7
18/12/2020	7236.7	7242.1	7209.7	440062500	1600	7219.5
19/12/2020	7219.5	7219.5	7162.8	624124300	1607.5	7179.2
.	.	.	.	.	.	.
.	.	.	.	.	.	.

Date is probably not a good predictor
(Although maybe day of the week is?)

Another example data set

	Predictors					Predictand
Date	FTSE Open	FTSE High	FTSE Low	FTSE Volume	Gold Close	FTSE Close
12/12/2020	7129.1	7168.6	7121.7	526735100	1567.4	7133.1
13/12/2020	7133.1	7205.1	7133.1	561646800	1575.1	7190.8
14/12/2020	7190.8	7232.8	7183	519107200	1582.7	7197
15/12/2020	7197	7261.6	7185.2	688422300	1585.8	7236.7
18/12/2020	7236.7	7242.1	7209.7	440062500	1600	7219.5
19/12/2020	7219.5	7219.5	7162.8	624124300	1607.5	7179.2
.	.	.	.	.	.	.

We can't use these values to predict FTSE close – it's cheating as we don't really have these values until the end of the day – by which time we know FTSE close anyway!

Another example data set

	Predictors all lagged by 1 day					Predictand
Date	FTSE Open	FTSE High	FTSE Low	FTSE Volume	Gold Close	FTSE Close
12/12/2020	.	.	.	.	.	7133.1
13/12/2020	7129.1	7168.6	7121.7	526735100	1567.4	7190.8
14/12/2020	7133.1	7205.1	7133.1	561646800	1575.1	7197
15/12/2020	7190.8	7232.8	7183	519107200	1582.7	7236.7
18/12/2020	7197	7261.6	7185.2	688422300	1585.8	7219.5
19/12/2020	7236.7	7242.1	7209.7	440062500	1600	7179.2
.	7219.5	7219.5	7162.8	624124300	1607.5	.

If I lag these values by one day – then I am using the previous day's value (which I have) to predict the next day's FTSE close
E.g. – if you look closely – FTSE Open for 12/12 is now aligned with FTSE Close for 13/12 – which is what I want – predicting one day ahead

Another example data set

	Predictors					Predictand
Date	FTSE Open	FTSE High	FTSE Low	FTSE Volume	Gold Close	FTSE Close
12/12/2020	.	.	.	.	.	7133.1
13/12/2020	7129.1	7168.6	7121.7	526735100	1567.4	7190.8
14/12/2020	7133.1	7205.1	7133.1	561646800	1575.1	7197
15/12/2020	7190.8	7232.8	7183	519107200	1582.7	7236.7
18/12/2020	7197	7261.6	7185.2	688422300	1585.8	7219.5
19/12/2020	7236.7	7242.1	7209.7	440062500	1600	7179.2
.	7219.5	7219.5	7162.8	624124300	1607.5	.

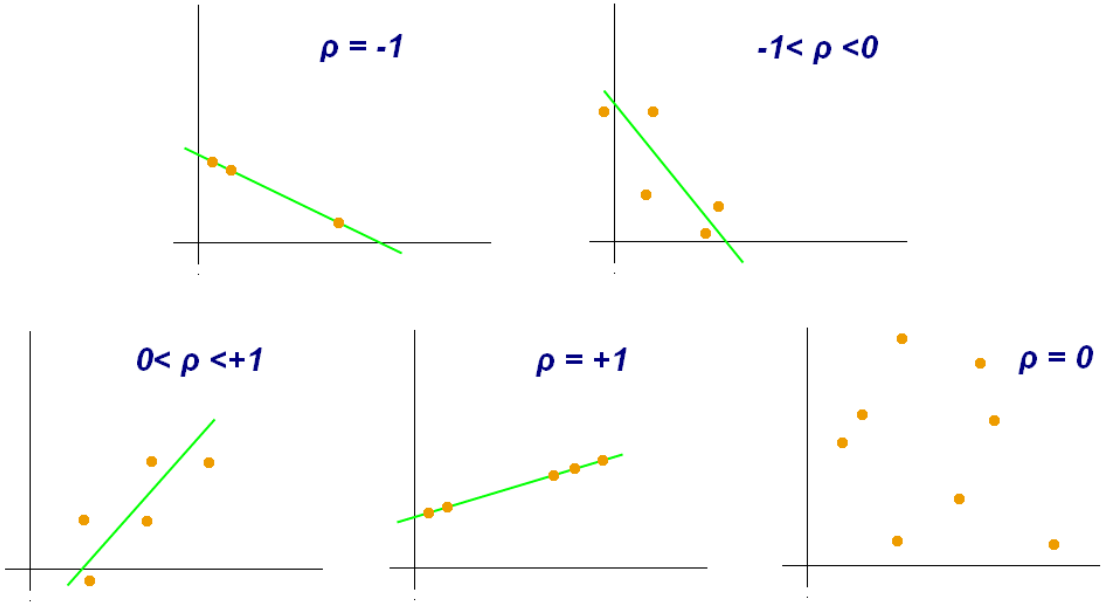
However, some predictors may correlate with *FTSE Close* better if I lag them for 2, 3, 4 or even more days (not just one day like I have here)

Explore the lagged correlations

- I would start to explore the correlation between each predictor and the predictand based on different lag times (t-1, t-2, t-3, t-4 . . . days before)
- E.g. we might find *Gold Close* three days before (t-3) has the strongest correlation with *FTSE Close* (better than any other lag time for *Gold*)
- We might find *FTSE Volume* one day before (t-1) has the strongest correlation with *FTSE Close*
- What I can't use is the same day's predictor to model that same day FTSE value – this is not creating model that is predicting the future – which is what I am after.

Correlation - interlude

- Correlation is a measure of the linear relationship between two variables.
- It ranges from -1 (perfect negative correlation), through zero (no relationship whatsoever) to +1 (perfect positive correlation)



[Wikipedia]

Correlation - interlude

- You can calculate the correlation coefficient r (if you want to program it) using:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Where:

x and y are our two variables to compare (n samples)

$\bar{x}$ is the mean of the x variable

$\bar{y}$ is the mean of the y variable

Or – use the correl(array1,array2) function in Excel

Another example data set

- I then shift each predictor an appropriate number of days so the strongest correlations are aligned with *FTSE Close*

Predictors					Predictand	
FTSE Open T-1	FTSE High T-2	FTSE Low T-1	FTSE Volume T-1	Gold Close T-3	Date	FTSE Close
.	.	.	.	.	12/12/2020	7133.1
7129.1	.	7121.7	526735100	.	13/12/2020	7190.8
7133.1	7168.6	7133.1	561646800	.	14/12/2020	7197
7190.8	7205.1	7183	519107200	1567.4	15/12/2020	7236.7
7197	7232.8	7185.2	688422300	1575.1	18/12/2020	7219.5
7236.7	7261.6	7209.7	440062500	1582.7	19/12/2020	7179.2
7219.5	7242.1	7162.8	624124300	1585.8	.	.
.	7219.5	.	.	1600	.	.

Another example data set

For example:

- This is the FTSE Open value for 12/12/2020
- It is now (aligned with) used to predict *FTSE Close* on 13/12/2020
- I.e. one day ahead

Predictors	Predictand	
FTSE Open T-1	Date	FTSE Close
.	12/12/2020	7133.1
7129.1	13/12/2020	7190.8
7133.1	14/12/2020	7197
7190.8	15/12/2020	7236.7
7197	18/12/2020	7219.5
7236.7	19/12/2020	7179.2
7219.5	.	.
.	.	.

Lagged Predictand

Note – because we are using time series data, we could also use a lag of our Predictand as a Predictor!

- This is the FTSE Close value for 12/12/2020
- It is now (aligned with) used to predict *FTSE Close* on 13/12/2020
- I.e. one day ahead
- For time series data – a lag of the predictand (t-1, t-2 etc) is usually a strong predictor for our model.

Predictors	Predictand	
FTSE Close T-1	Date	FTSE Close
.	12/12/2020	7133.1
7133.1	13/12/2020	7190.8
7190.8	14/12/2020	7197
7197	15/12/2020	7236.7
7236.7	18/12/2020	7219.5
7219.5	19/12/2020	7179.2
7179.2	.	.
.	.	.

Another example data set

- You would then delete any rows with missing values to give you your full data set (it doesn't matter what order the predictor columns are in)

Predictors						Predictand	
FTSE Close T-1	FTSE Open T-1	FTSE High T-2	FTSE Low T-1	FTSE Volume T-1	Gold Close T-3	Date	FTSE Close
.	.	.	.	.	.	.	.
7197	7190.8	7205.1	7183	519107200	1567.4	15/12/2020	7236.7
7236.7	7197	7232.8	7185.2	688422300	1575.1	18/12/2020	7219.5
7219.5	7236.7	7261.6	7209.7	440062500	1582.7	19/12/2020	7179.2
	.	.	.	.	.	.	.

(I have added a lagged predictand to the predictors here)

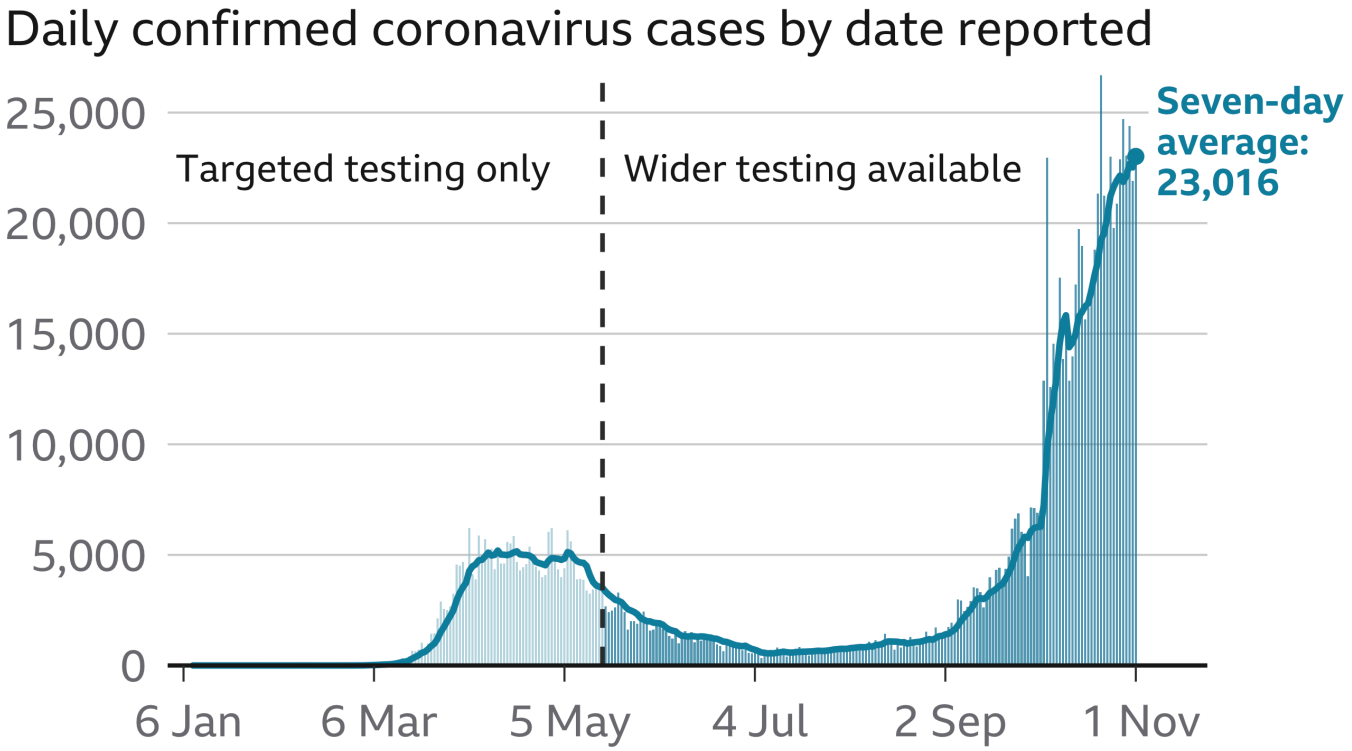
You would then standardise and the split the data set as before.

Moving averages

- Moving averages provide a means of smoothing out peaks (outliers) in time series data
- They also provide a means of combining the information from several previous time steps into a single predictand
- Often used in financial forecasting, hydrological forecasting etc

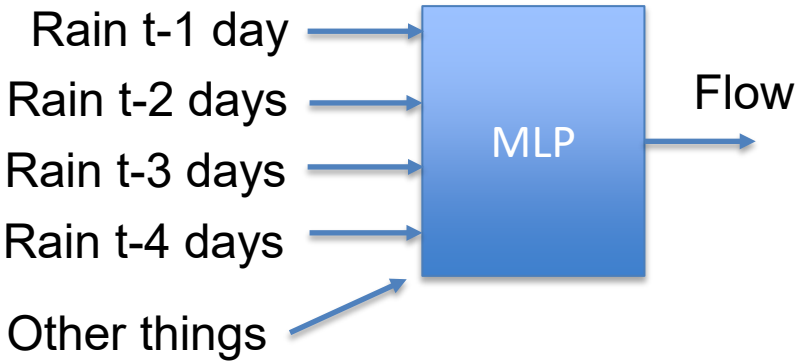
Moving Average

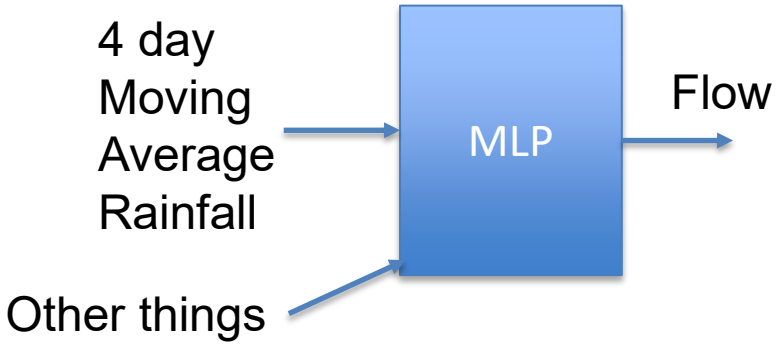
Moving Average used to smooth out blips



Moving Average

- Consider river flow
- Probably based on rainfall over the past few days
- Rather than having an input for each of the past few days – let us use a moving average instead to combine the days together





Moving Average - example

e.g.

$$MA_2 = \frac{R_t + R_{t-1}}{2}$$

MA_2: Moving Average of 2

R_t: Data at current step

R_{t-1}: Data at one step back

Time step	Raw Data	Moving Average 2 Steps	Moving Average 3 Steps	Moving Average 4 steps
1	20	-	-	-
2	30	25.0	-	-
3	40	35.0	30.0	-
4	32	36.0	34.0	30.5
5	54	43.0	42.0	39.0
6	33	43.5	39.7	39.8
7	2	17.5	29.7	30.3
8	33	17.5	22.7	30.5
9	45	39.0	26.7	28.3
10	58	51.5	45.3	34.5

Weighted Moving Average (WMA)

- Weights the values used in the averaging
- EG – rainfall yesterday more influential than rainfall two days ago – so give this more weight in the MA calculation
- E.g. $$WMA_3 = \frac{0.5R_{t-1}+0.3R_{t-2}+0.2R_{t-3}}{3}$$

R_{t-n} rainfall n days ago

Weighted Moving Average

- Can have moving averages over as many time steps as you wish

- For weighted moving averages the sum of the weights must be one – otherwise we end up with strange values

$$WMA_3 = \frac{0.5R_{t-1}+0.3R_{t-2}+0.2R_{t-3}}{3}$$

(weights : 0.5+0.3+0.2=1)

Lagged Moving Averages

- It might also be the case that a lagged moving average (t-1, t-2, t-3, etc) might provided a better predictor than an unlagged MA.

- You can also consider exploring lags of your moving averages as potential predictors.

Creating and training ANNs

- In a nutshell:
  - Process the data ✓
  - **Train the network(s)**
  - Assess the results

Backpropagation – see separate slides

- There are numerous ways to train ANNs
- Depends on the ANN we have chosen
- Examples – Adaline, Madaline, Boltzmann Learning, K-means clustering, Sammon's Projection, Associative Memory Learning, Conjugate Gradients (and variations), etc.
- We are going to look at the most common means of training MLPs to start with – ***error backpropagation***

- ***Question – why it is a good idea to code up your own MLP?***

Creating and training ANNs

- In a nutshell:
  - Process the data ✓
  - Train the network(s) ✓
  - **Assess the results**

Real Life Example

River level

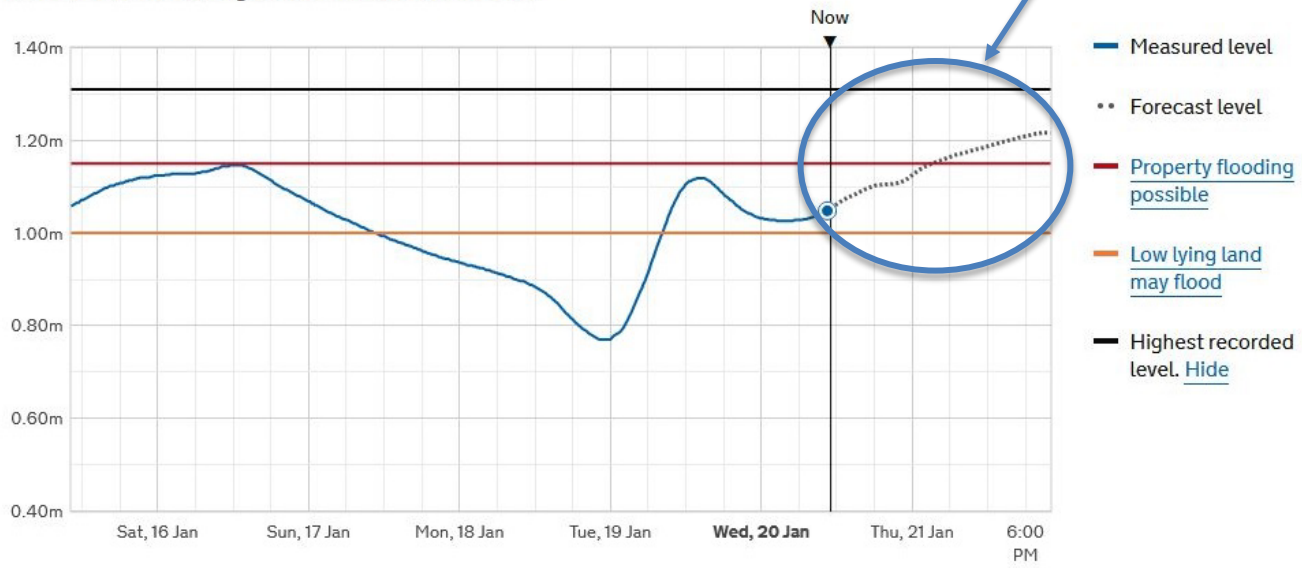
River Soar at Kegworth

A flood warning is in place nearby.

Flooding is expected. Immediate action required. View details.

Latest recorded level **1.05m** at **10:30am Wednesday 20 January 2021**.

Levels: last 5 days and next 36 hours



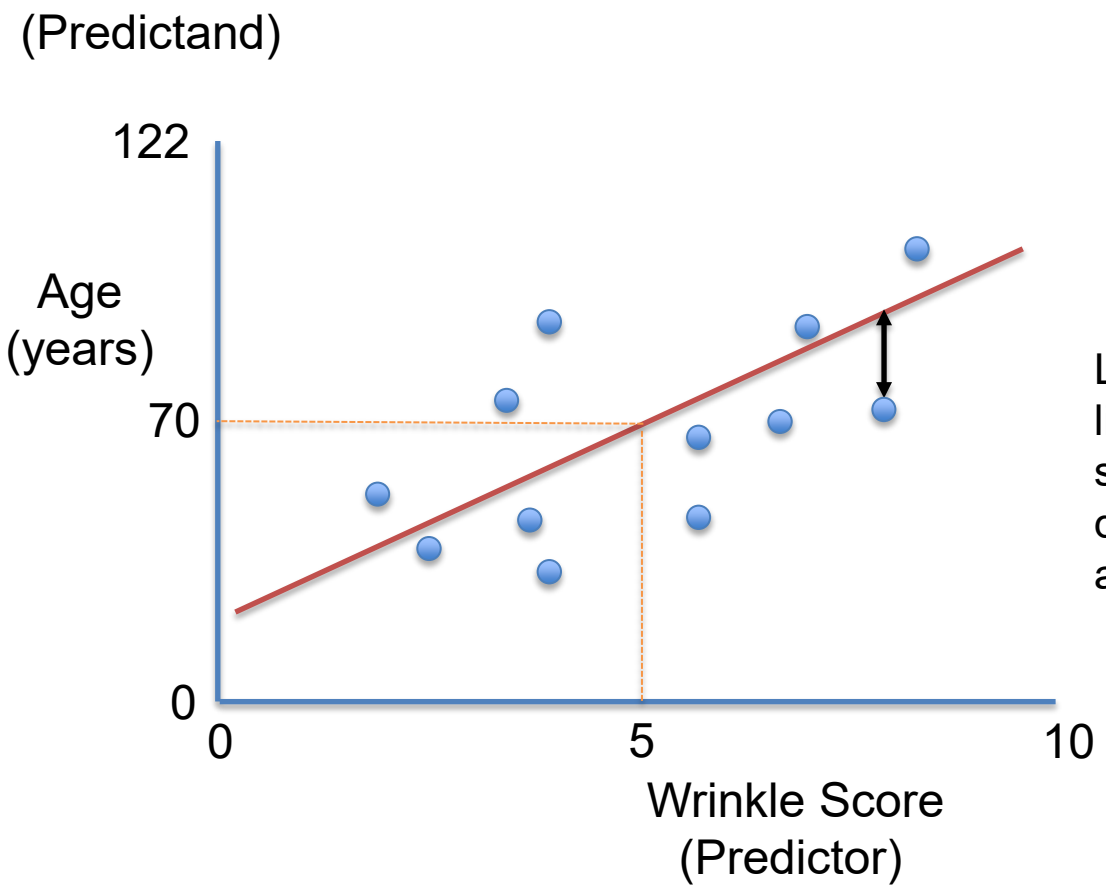
Environment Agency Prediction (modelled)

Loughborough University

Model Assessment

- When I evaluate my trained ANN model – it is useful to compare it with a simple baseline equivalent
- What is the point of me developing a complex ANN model when a simple model might give me pretty much the same accuracy of results?
- We can compare our ANN with a simple statistical model (eg a linear regression model); or with a simple statistical value (eg mean or moving average, etc)

Simple linear regression – as a baseline



Sample data point
(a person)
There are 12 people
in our data set

Linear regression finds a straight line through the data such that the sum of the squared distances between the line and all the points is minimised

We can then use this line to make a prediction -
E.g. A Wrinkle Score 5 gives a predicted age of 70 years

(Multiple Linear Regression – lots of predictors)

Model Assessment – example baseline

- Say I want to predict the outcome of a football match (home win, draw, or away win)
- If you examine all the Premiership scores over several seasons; around 47% are home wins; 26% are draws; 27% are away wins

- A simple model in this case –
- Simply predict every match as a home win and I am right 47% of the time!

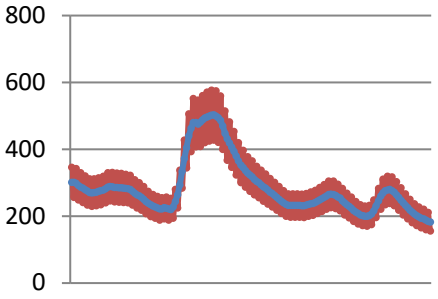
Model Assessment – example baseline

- So – if we develop an ANN to model football scores it needs to do a lot better than 47% accuracy – otherwise there is no point.

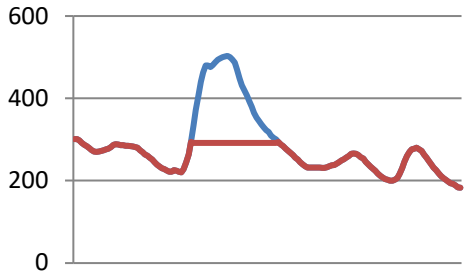
- Other examples of baselines we want to beat –

- Predicting FTSE 100 – average of last 5 days; or close of previous day
- Predicting today's temperature – use yesterday's
- Predicting crop yield – use last year's value

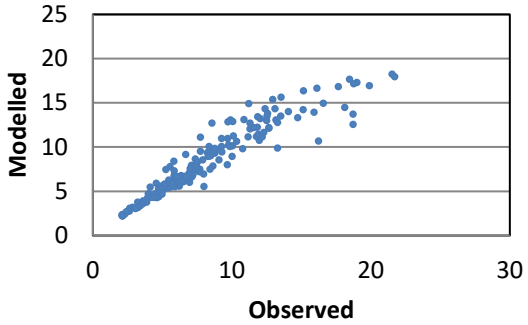
Model Assessment

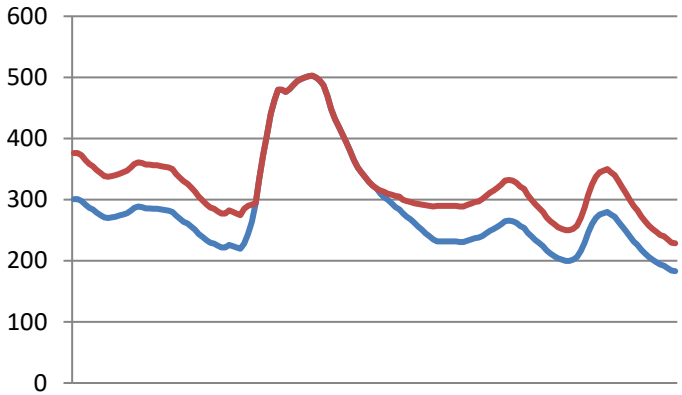


Noisy

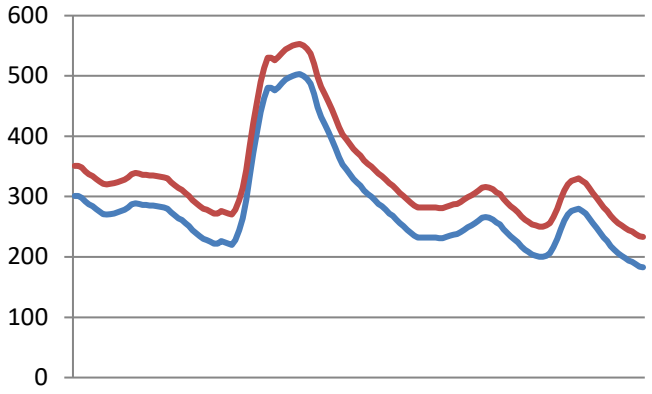


Low





High



Raised

Loughborough University

Model Assessment

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}\left(\hat{Q}_i - Q_i\right)^2}{n}}$$

$$MSRE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\hat{Q}_i - Q_i}{Q_i}\right)^2$$

$$CE = 1 - \frac{\sum_{i=1}^{n}\left(\hat{Q}_i - Q_i\right)^2}{\sum_{i=1}^{n}\left(Q_i - \overline{Q}\right)^2}$$

$$RSqr = \left(\frac{\sum_{i=1}^{n}(Q_i - \bar{Q})(\hat{Q}_i - \tilde{Q})}{\sqrt{\sum_{i=1}^{n}(Q_i - \bar{Q})^2 \sum_{i=1}^{n}(\hat{Q}_i - \tilde{Q})^2}}\right)^2$$

Q_i is the observed value

$\hat{Q}_i$ is the modelled value

$\bar{Q}_i$ is the mean of the observed values

$\tilde{Q}_i$ is the mean of the modelled values

RMSE – Root Mean Squared Error

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}\left(\hat{Q}_i - Q_i\right)^2}{n}}$$

- Records in real units the level of overall agreement between the observed and modelled datasets.
- Positive, no upper bound
- For a perfect model the result would be zero.

MSRE –Mean Squared Relative Error

$$MSRE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\hat{Q}_i - Q_i}{Q_i}\right)^2$$

- Error is **relative** to the observed value.
- Eg 101 vs 102 compared with 1 vs 2

Both have same absolute error but relative error in first case is much smaller (I would be happy if my model produced 101 when the observed value was 102; not so happy with 1 when the observed is 2)

- Positive with no upper bound - for a perfect model the result would be zero.

CE – Coefficient of Efficiency

- CE is intended to range from 0 to 1 but negative scores are also permitted.
- +1 represents a perfect model;

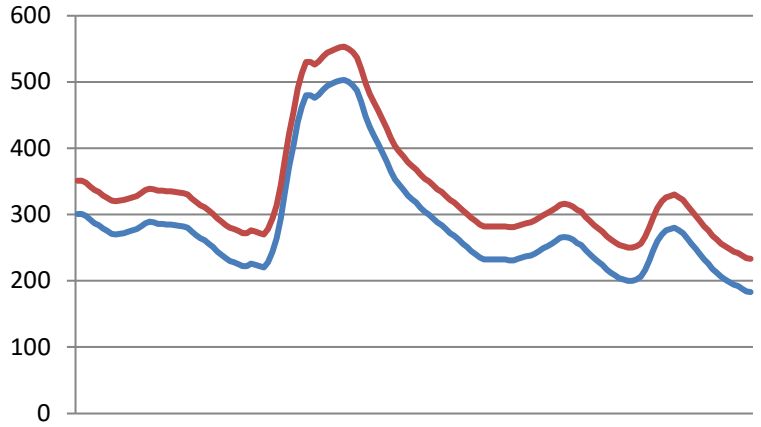
$$CE = 1 - \frac{\sum_{i=1}^{n}\left(\hat{Q}_i - Q_i\right)^2}{\sum_{i=1}^{n}\left(Q_i - \overline{Q}\right)^2}$$

- zero indicates that the model is no better than "no knowledge" model - forecast is the mean of the observed data;
- Negative scores - worse that a "no knowledge" model i.e. rubbish!.

RSqr – R-Squared (Coefficient of Determination)

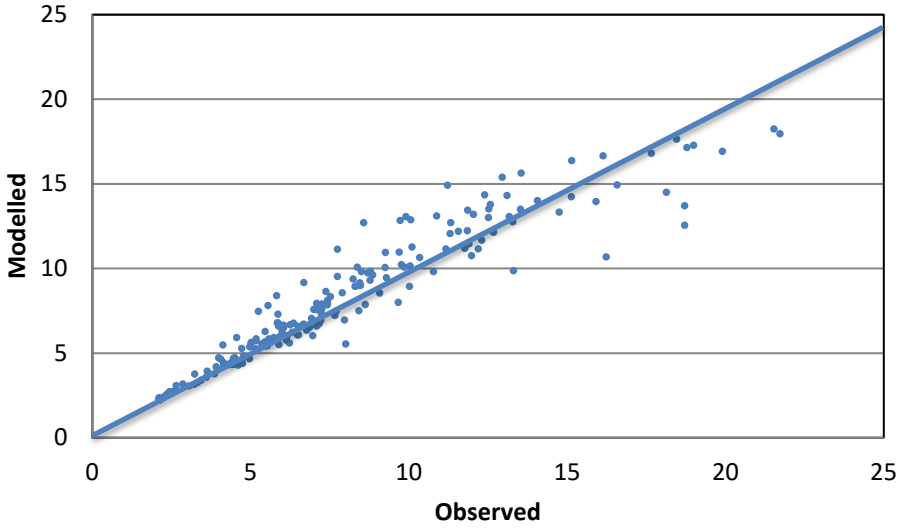
- Squared ratio of the combined dispersion of the two series to the total dispersion of the observed and modelled series.
- Measures the coincidence of the shape!
- Ranges from 0 (rubbish!) to 1 (perfect fit)

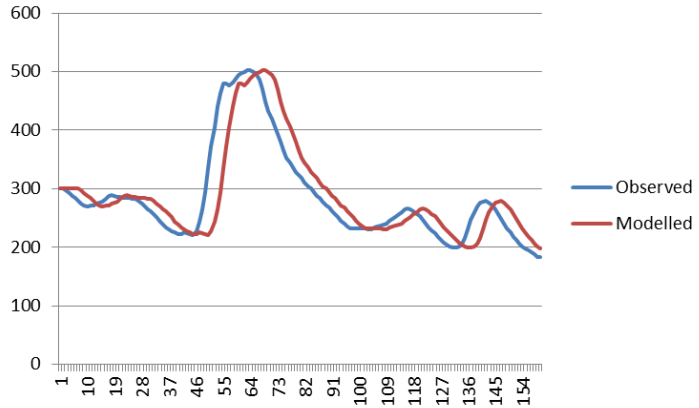
$$RSqr = \left(\frac{\sum_{i=1}^{n}(Q_i - \bar{Q})(\hat{Q}_i - \tilde{Q})}{\sqrt{\sum_{i=1}^{n}(Q_i - \bar{Q})^2 \sum_{i=1}^{n}(\hat{Q}_i - \tilde{Q})^2}} \right)^2$$



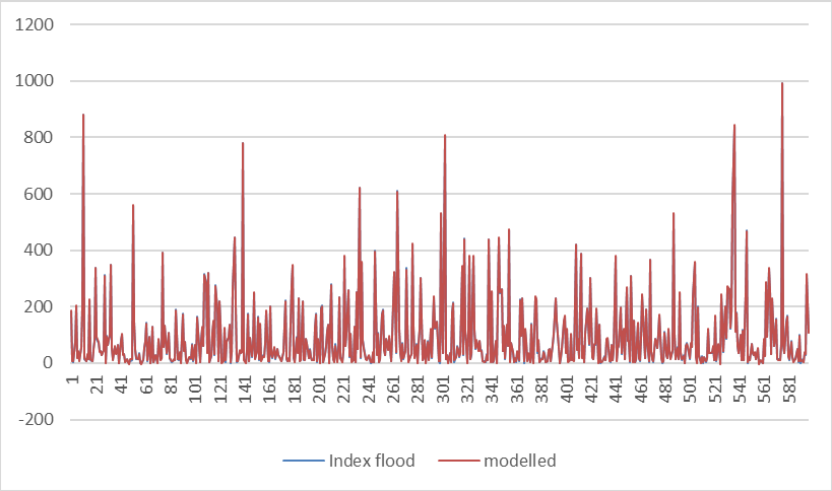
Error measures

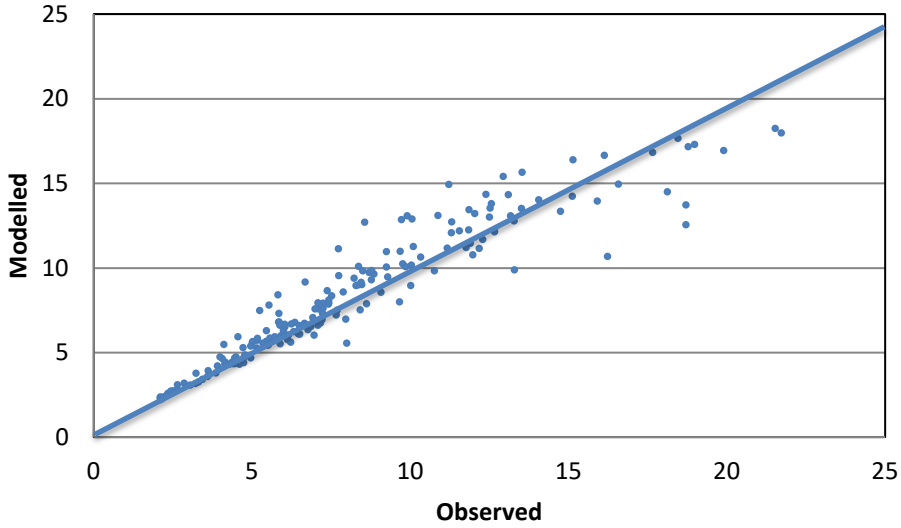
- Plot the results!





Plots





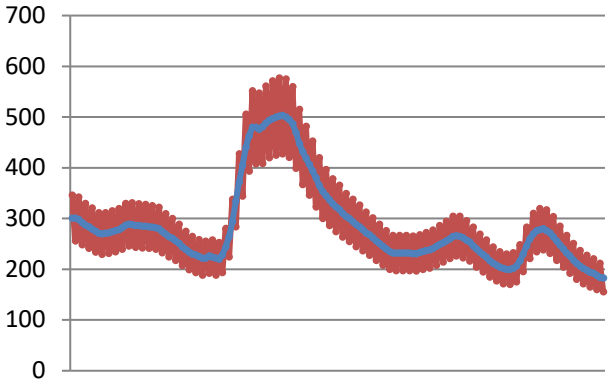
Line charts only make sense with time series data

Better to use a scatter plot otherwise

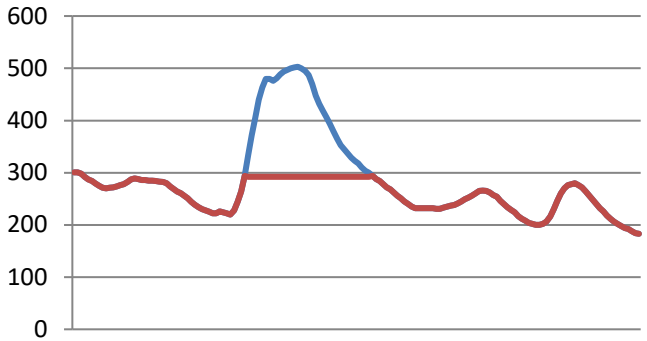
The closer the dots to the line – better the model

can also spot persistent over/under estimation and perhaps other issues – e.g. worse errors for higher values etc.

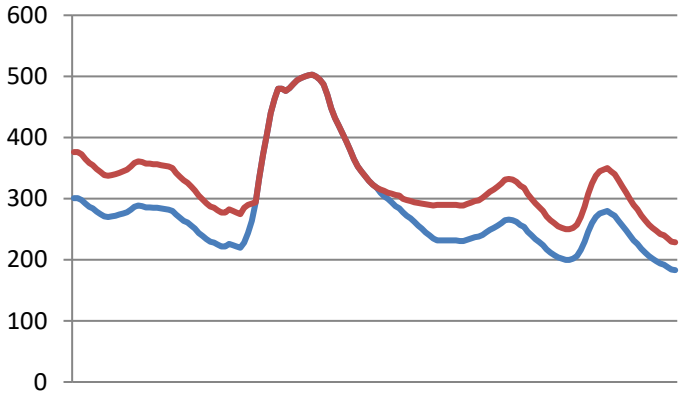
Performance measures



Noisy

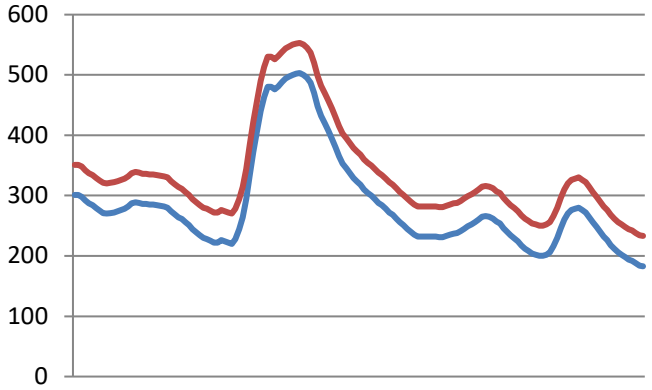


Low



High

Which model is 'best'?



Raised

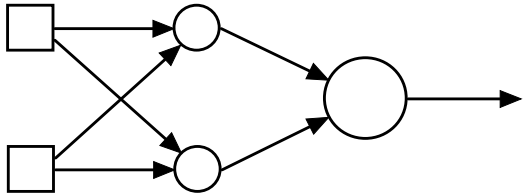
Performance measures

	RMSE	MSRE	CE	R-Squared
Low Flow	67.48	**0.0208**	0.3151	0.5604
High Flow	52.55	0.0447	0.5847	0.9240
Raised	51.04	0.0391	0.6082	**0.9598**
Noisy	**44.58**	0.0225	**0.7010**	0.7704

www.hydrotest.org.uk

Category Classification

- Output [0,1]
  - O<0.5 = False
  - O>=0.5 = True



- Output [-1, +1]
  - O<-0.2 = Low Flood Risk
  - -0.2<O<0.2=Medium Flood Risk
  - O>0.2=High Flood Risk

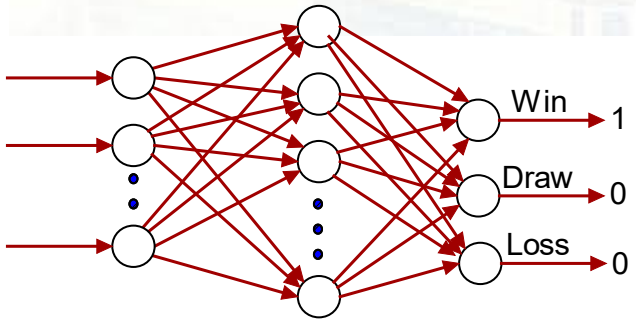
Category Classification

- Football results:

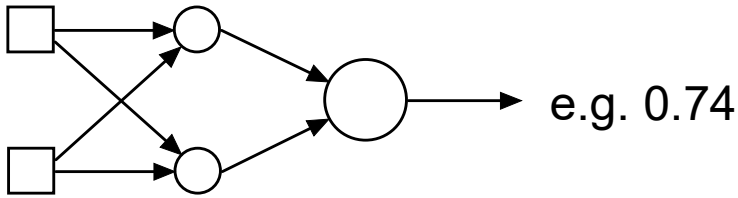
0-0, 1-1, etc = Draw;

1-0, 2-0, 2-1, etc. = Win;

0-1, 0-2, 1-2, etc. = Loss



- Single output [W,D,L]
  - O<-0.2 = Loss
  - -0.2<=O<=0.2 = Draw
  - O>0.2 = Win



Category Classification – confusion matrix

	Actual	
Predicted	True	False
True	A ✓	B ✗
False	C ✗	D ✓

We want lots of As and Ds; zero Bs and Cs

A+B+C+D = total data = N

Correct classification rate = (A+D)/N
Misclassification rate = (B+C)/N
False positive rate = B/(B+D)
False negative rate = C/(A+C)

You could report these values in a summary table to compare models

Presenting results – some points

- Do not go crazy with decimal places – eg reporting results of 0.7432940234
- Make sure all axes are labelled in graphs
- Make sure correct units are displayed (eg if a measure is in mm or £ make sure this is shown)
- Tables are a good way to summarize lots of results
- Graphs are a good way to compare models – can spot things in a graph that is not always clear from statistics alone

Creating and training ANNs

- In a nutshell:
  - Process the data ✓
  - Train the network(s) ✓
  - Assess the results ✓

Artificial Neural Networks

- Introduction ✓
- Machine Learning ✓
- History ✓
- Creating/Training ANNs ✓
- Data processing ✓
- Evaluation ✓
- **A few things to finish**

A few things to finish

- **Advantages and disadvantages**
- Application Areas
- Recent Research
- Tools

Advantages of ANNs

- **New problems** - ANNs are well suited to new problems that are difficult to define. They act as 'black boxes'.

- We don't need to fully understand the problem to apply them (but this is, perhaps, a disadvantage)

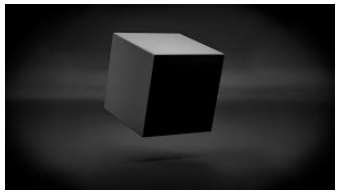
- **Robustness** - ANNs can handle missing and fuzzy data. Because data and processing is distributed throughout an ANN they can tolerate faults and can tolerate damage to themselves.

Advantages of ANNs

- **Fast processing** - can solve complex problems quickly once trained by operating on problems using a massively interconnected number of processing units.

- **Flexibility** - can adapt to changing environments. Easy to maintain and can learn new things.

Summary of criticisms of ANNs



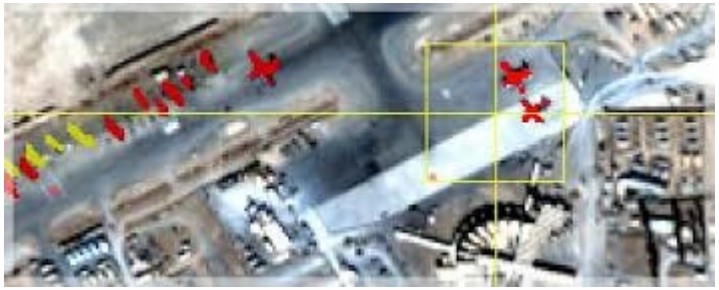
- No physical reasoning/ explanation (i.e., black boxes)
- Inability to generalise to extreme events outside training data
- No single “true” solution (i.e., equifinality)
- Difficult to assign confidence limits
- Over–parameterised
- Fails to build on conventional “wisdom” - ‘Thrown’ at problems without thought

A few things to finish

- Advantages and disadvantages 
- **Application Areas**
- Recent Research
- Tools

Application Areas - Image processing

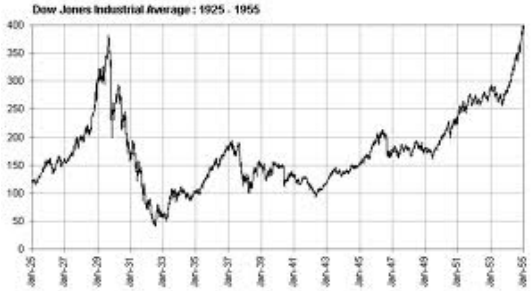
- Many application areas – image identification, feature extraction, data reduction, image segmentation, pre-processing/filtering etc.

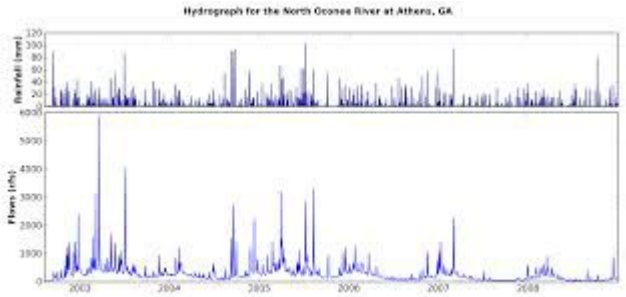


Deep learning ANNs used in these areas

Application Areas - Prediction/forecasting

- Using a set of samples (training patterns) to build a model to predict future results
- Examples – flood forecasting, stock market prediction, weather forecasting, crop yields, etc.



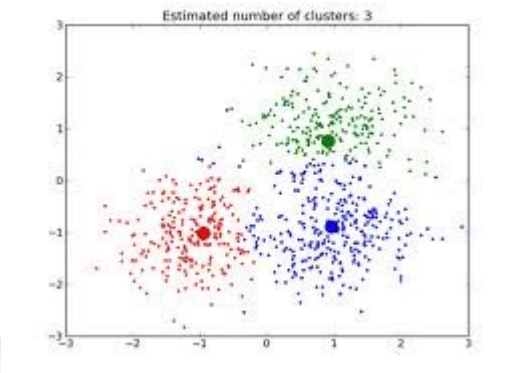


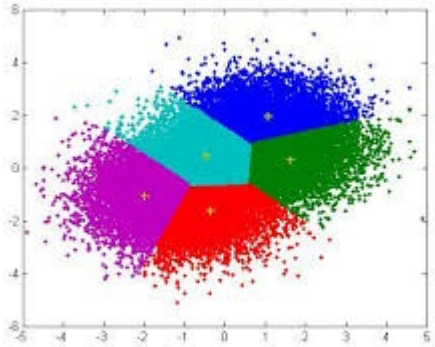
Application Areas - Optimisation

- Lots of problems in maths, science, economics require optimised solutions.
- Find the best solution from a series of feasible solutions
- Examples – travelling salesman problem, game strategy, chess, economics, computer networks, site location, etc.

Application Areas - Clustering

- Unsupervised pattern classification
- Training data with unknown classes
- ANN explores similarities in patterns and clusters them
- Examples – data mining, data compression, exploratory data analysis





A few things to finish

- Advantages and disadvantages 
- Application Areas 
- **Recent Research**
- Tools

Recent research

- Assessment of network limitations and generalisation abilities
- Deep learning
- Fusion with other technologies - Genetic Algorithms, Fuzzy approaches
- Implementation of ANNs using dedicated hardware
- Interpretation of the black box and sensitivity analysis

A few things to finish

- Advantages and disadvantages 
- Application Areas 
- Recent Research 
- **Tools**

Tools

- Weka
- Tensor Flow
- Microsoft Azure
- AWS – Amazon Web Services
- Matlab
- Python

Weka

- ‘Weka is a collection of machine learning algorithms for data mining tasks.
- It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization’
- Supports deep learning
- Free!

Tensor Flow

- Helps you develop and train ML models.
- Includes Deep learning.
- Developed by Google
- Free!

Microsoft Azure

- Azure is Microsoft's cloud computing services
- Can be used to build, train and deploy various ML models
- Includes deep learning, supervised, unsupervised learning models

AWS – Amazon Web Services

- Similar to Microsoft Azure
- Provides ML and AI services for businesses
- Has pre-trained services or you can develop your own
- For – Computer Vision, forecasting, language processing etc.

Matlab and Python

- **MATLAB** (*matrix laboratory*) is a numerical computing environment
- It has libraries to enable you to develop ANNs

- **Python -** High level programming language
- Has classes that allow you to create ANNs

A few things to finish

- Advantages and disadvantages 
- Application Areas 
- Recent Research 
- Tools 

Questions?

Handy definitions

- ANN – Artificial Neural Network
- CNN – Convolution Neural Network
- DL – Deep Learning
- ML – Machine Learning
- MLP – Multi-Layer Perceptron
- RNN – Recurrent Neural Network