UNIVERSITY OF PORTSMOUTH

Intro to Machine Learning

Chris Frohmaier

Thanks to: Becky Canning

Institute of Cosmology and Gravitation

What is AI and ML?

- All machine learning is AI, but not all AI is machine learning
- Artificial Intelligence is a catch-all term for any program that can turn information into a decision without active human input.

- In the most basic form, it is a series of if-then-else statements
  - e.g. `IF temperature<16 THEN turn_on_heating`

- The central heating will then work with further input.
  - But we can all agree this is rudimentary!

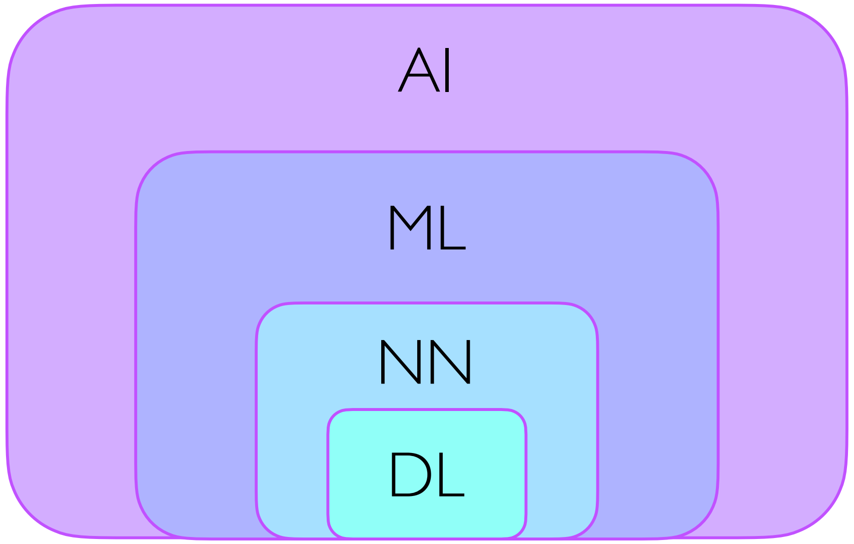
A brief history

- 1943 - A mathematical model for neurons
- 1950 - The Turing Test - Can machines think?
- 1955 - First mention of 'AI'
- 1960's - Expert systems and symbolic AI
- 1960's - First connected neuron system in code but without an optimiser
- 1986 - First learning algorithm for connected neuron systems
- 1998 - First Convolutional Neural Network

https://theconversation.com/a-brief-history-of-ai-how-we-got-here-and-where-we-are-going-233482
https://courses.cs.umbc.edu/471/papers/turing.pdf

The Jargon (as we understand it today)

Artificial Intelligence: The science or engineering discipline which deals with machines which can mimic human cognition, decision making or perception.



UNIVERSITY OF PORTSMOUTH

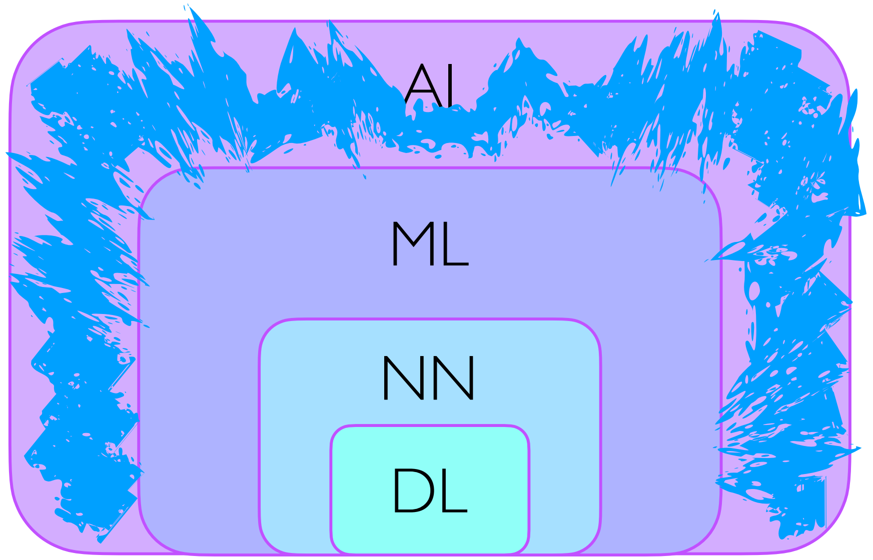
The Jargon (as we understand it today)

Artificial Intelligence: The science or engineering discipline which deals with machines which can mimic human cognition, decision making or perception.

Any 'rules' statements

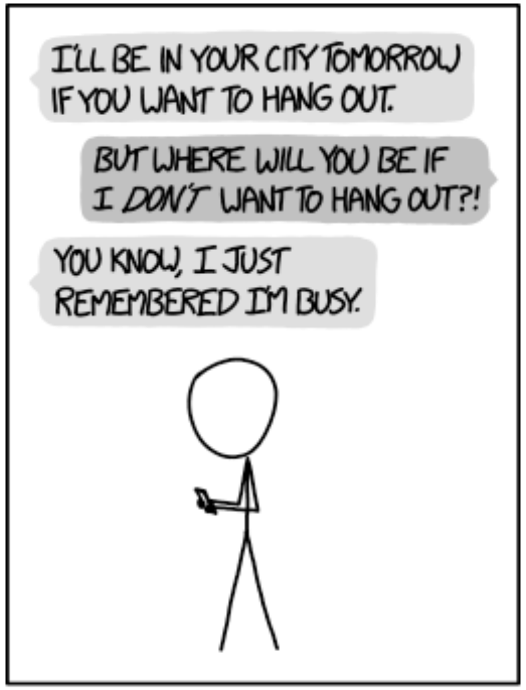
If … then … statements

Symbolic AI



Knowledge graphs

Expert Systems


I'LL BE IN YOUR CITY TOMORROW IF YOU WANT TO HANG OUT.
BUT WHERE WILL YOU BE IF I DON'T WANT TO HANG OUT?!
YOU KNOW, I JUST REMEMBERED I'M BUSY.

WHY I TRY NOT TO BE PEDANTIC ABOUT CONDITIONALS.

And others …

Good Old Fashioned AI (GOFAI) → **Human hard codes rules in code**

Preconceptions are hard coded into the machine

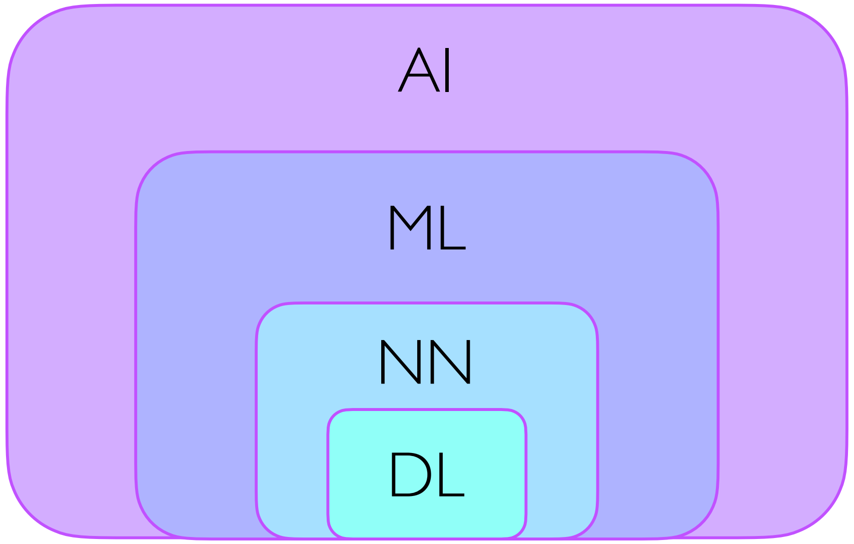
How do we update the rules when we learn something new?

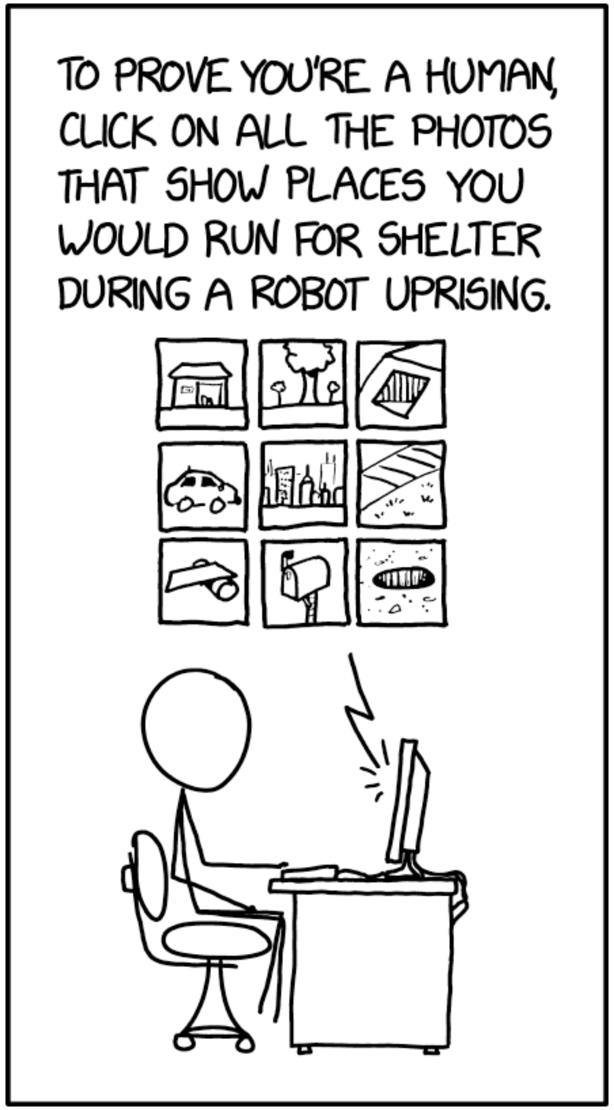
UNIVERSITY OF PORTSMOUTH

The Jargon (as we understand it today)

Machine Learning: Algorithms that can learn from data to produce an inference which was not explicitly coded.

Preconceptions still exists: Machines might not be told explicitly what to learn but assumptions are still built in as the machine is told how to learn and/or what to learn from.




TO PROVE YOU'RE A HUMAN, CLICK ON ALL THE PHOTOS THAT SHOW PLACES YOU WOULD RUN FOR SHELTER DURING A ROBOT UPRISING.

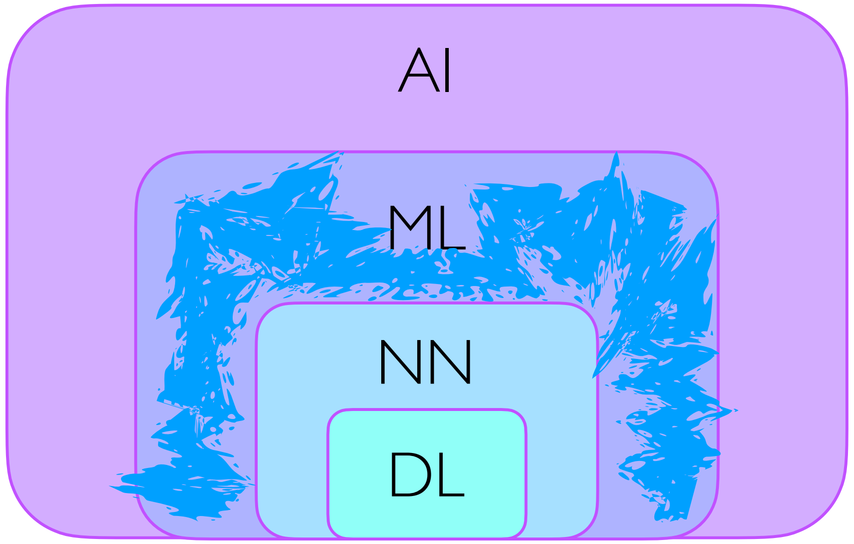
The Jargon (as we understand it today)

Machine Learning: Algorithms that can learn from data to produce an inference which was not explicitly coded.

Bayesian non-linear regression

Bayesian Networks

K-means

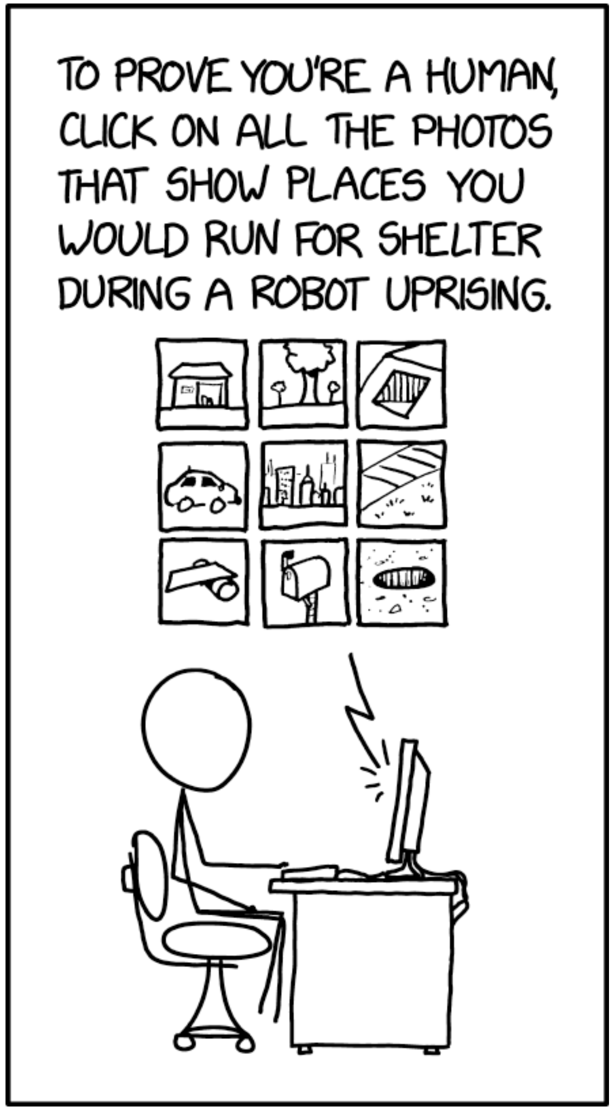


Linear regression

Decision trees

Hidden Markov Models

Support vector machines


TO PROVE YOU'RE A HUMAN, CLICK ON ALL THE PHOTOS THAT SHOW PLACES YOU WOULD RUN FOR SHELTER DURING A ROBOT UPRISING.

And others …

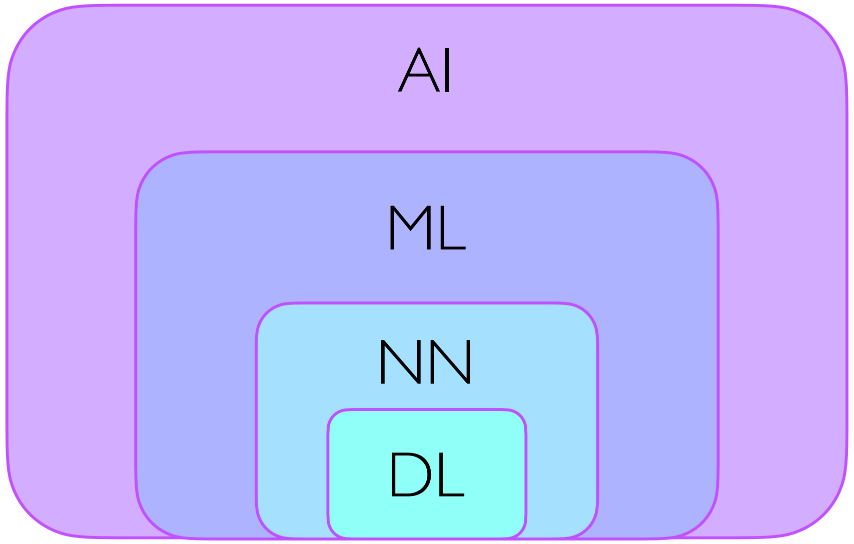
'Traditional ML' → Great with good quality structured and labelled data

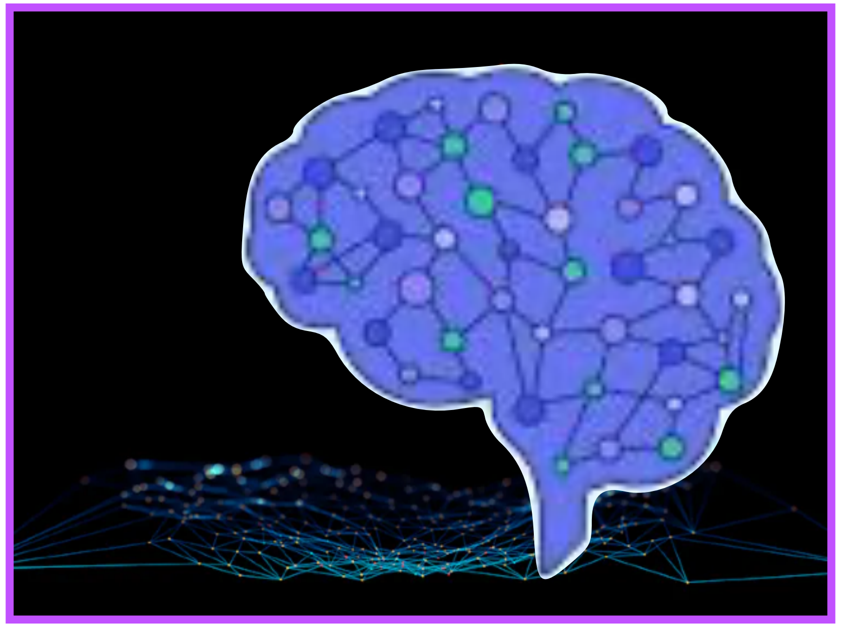
Unstructured data might need the freedom of NN

UNIVERSITY OF PORTSMOUTH

The Jargon (as we understand it today)

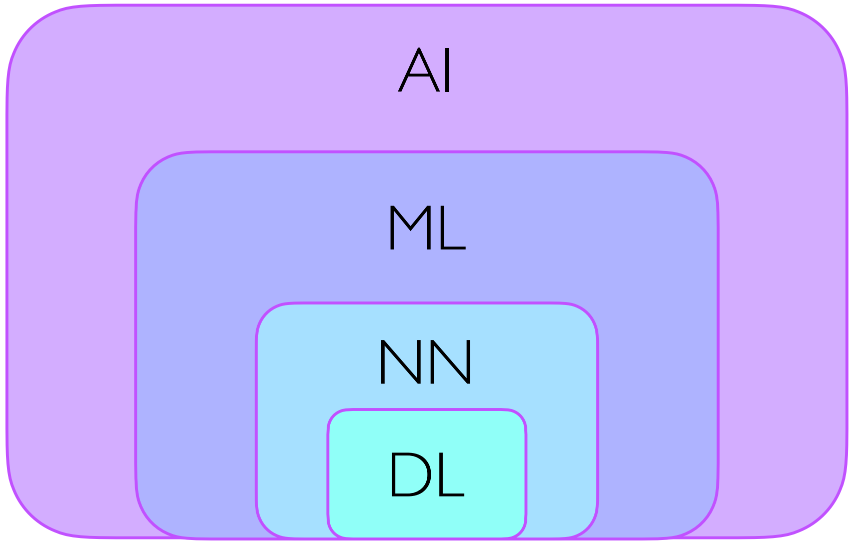
Neural Network: A ML approach which is loosely modelled on the human brain by using layers of connected 'nodes'.





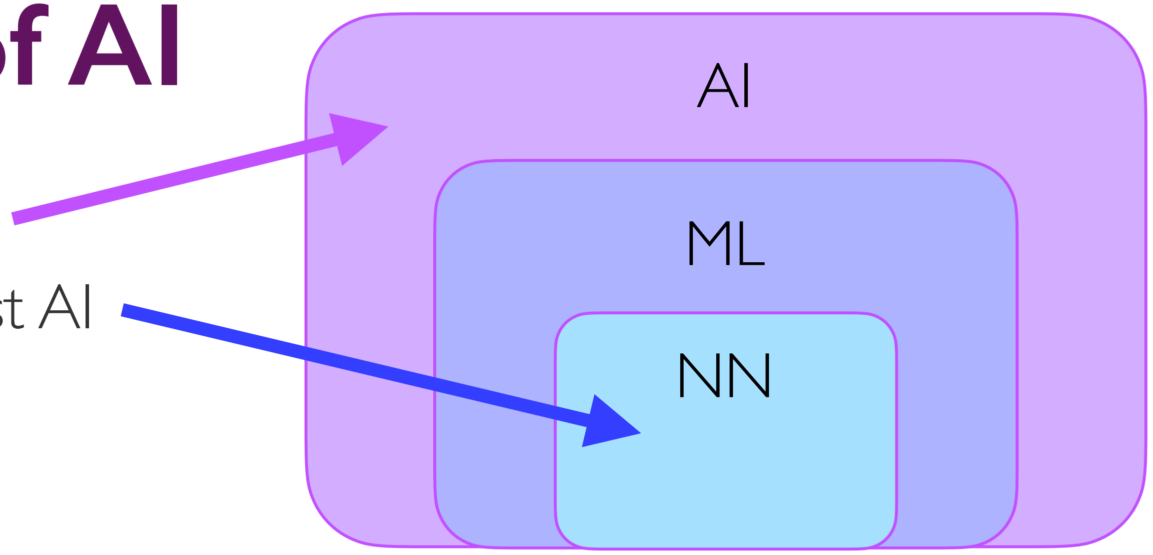
The Jargon (as we understand it today)

Deep Learning: Learning based on the use of a deep (many layered) neural network.



Types of AI

- Symbolic AI
- Connectionist AI

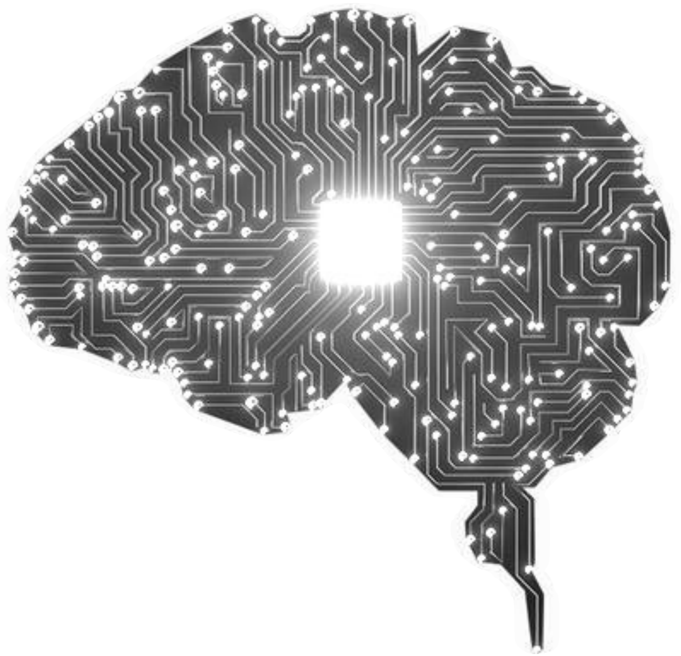


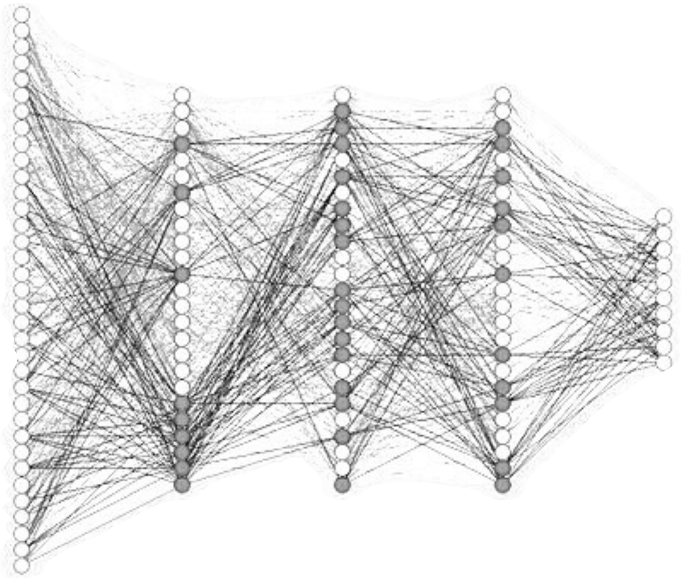
Symbolic AI (or GOFAI): Symbols used as a representation of knowledge; symbols are then connected through list and networks which have hardcoded rules (essentially if/then statements).
* Good when you have clear rules
* Good when you want code to be easily traceable
* Not great when there are huge numbers of rules which would be needed
* Not great when you don't know all the rules to code

Connectionist AI: A network of interconnected layers of nodes is optimised to form an inference.
* Good when you have loads of great training data
* Not easily understandable

NOTE: THIS IS NOT ALL THERE IS! You can blend all these techniques (and those mentioned before)
Explainable AI and Neuro-Symbolic AI (and probably other jargon filled words with 'AI' after it): Blend the two in an attempt to make AI more explainable, for example using one NN to find patterns in limited circumstances and another trained on question-answer pairs about the first ones classifications such that a human could 'query' the first algorithm.

Types of AI (Other terms you might hear)





Strong AI:
* Artificial General Intelligence
* Artificial Super Intelligence

Weak AI:
* Artificial Narrow Intelligence

This one doesn't really exist…

How do the Machines Learn?

- Types of learning
  - Supervised
  - Semi-supervised
  - Unsupervised
  - Reinforcement

- What will be used in this course?

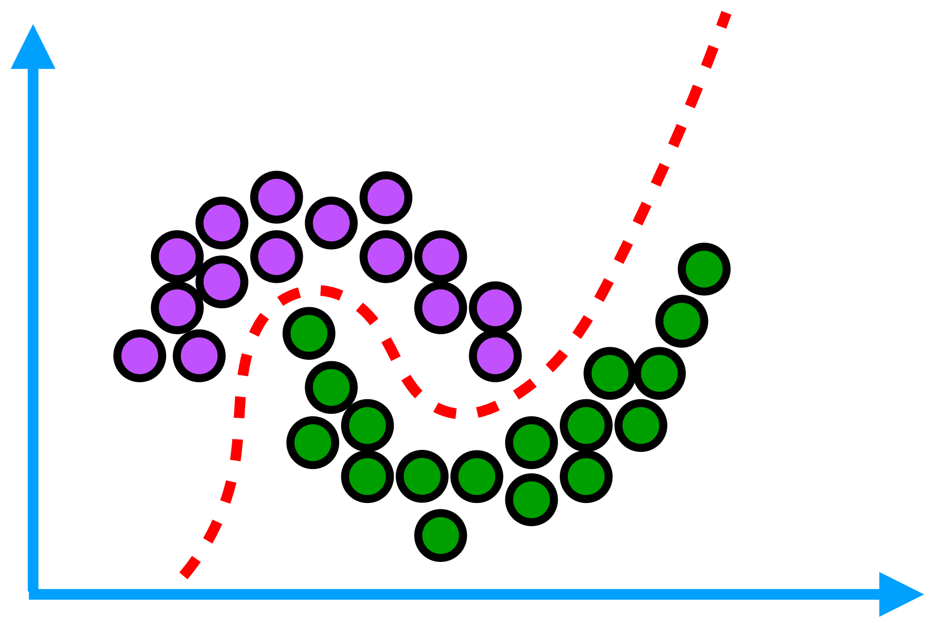
UNIVERSITY OF PORTSMOUTH

Learning Cheat Sheet

Supervised	Semi-supervised	Unsupervised	Reinforcement
Fully Labelled Learning Data	Hybrid Labelled and Unlabelled Data	Unlabelled Learning Data	Reward upon 'final action'
Machine compares it's guess to the labelled values	Machine treats loss for labelled and unlabelled data separately	Machine compares it's guess to some properties of the data e.g. it's distribution	Machine learns via trial and error and maximising reward functions.
Regression / Classification	Regression / Classification / Clustering	Clustering	Path optimisation

UNIVERSITY OF PORTSMOUTH

Supervised

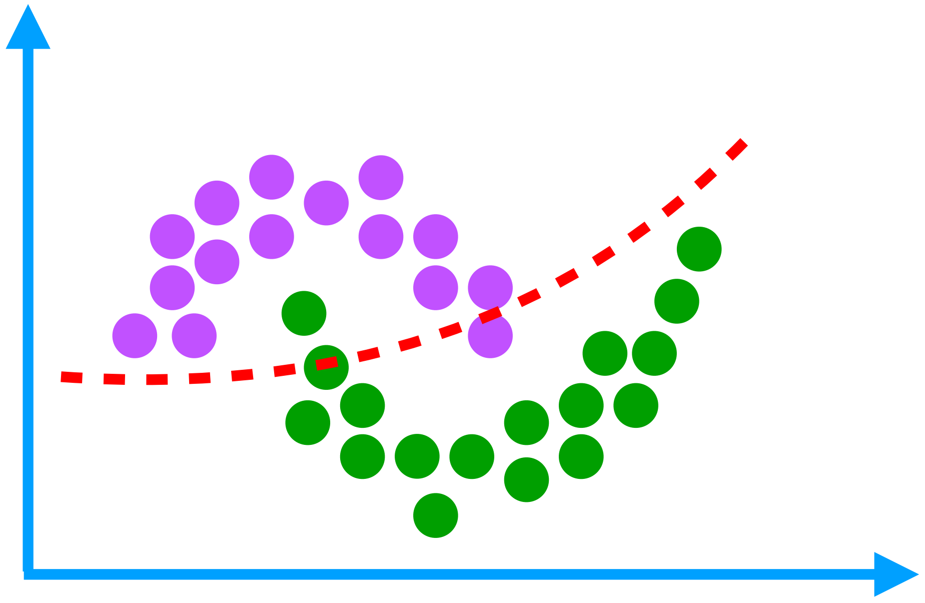


Example with binary classification

- Two 'moons' datasets (purple and green circles)
- The datasets are all labelled (black circles)
- The algorithm will first initialise a random guess and test against the correct values
- With all data labelled it will quickly learn to draw a decision boundary for the classes

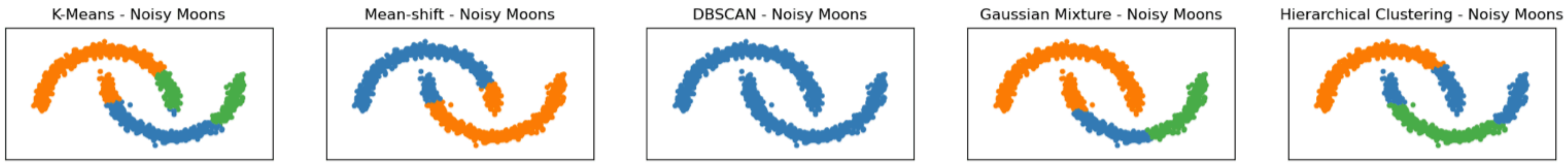
Guess → Compare with labels → adjust guess → repeat until metric reached

Unsupervised



Example with binary classification

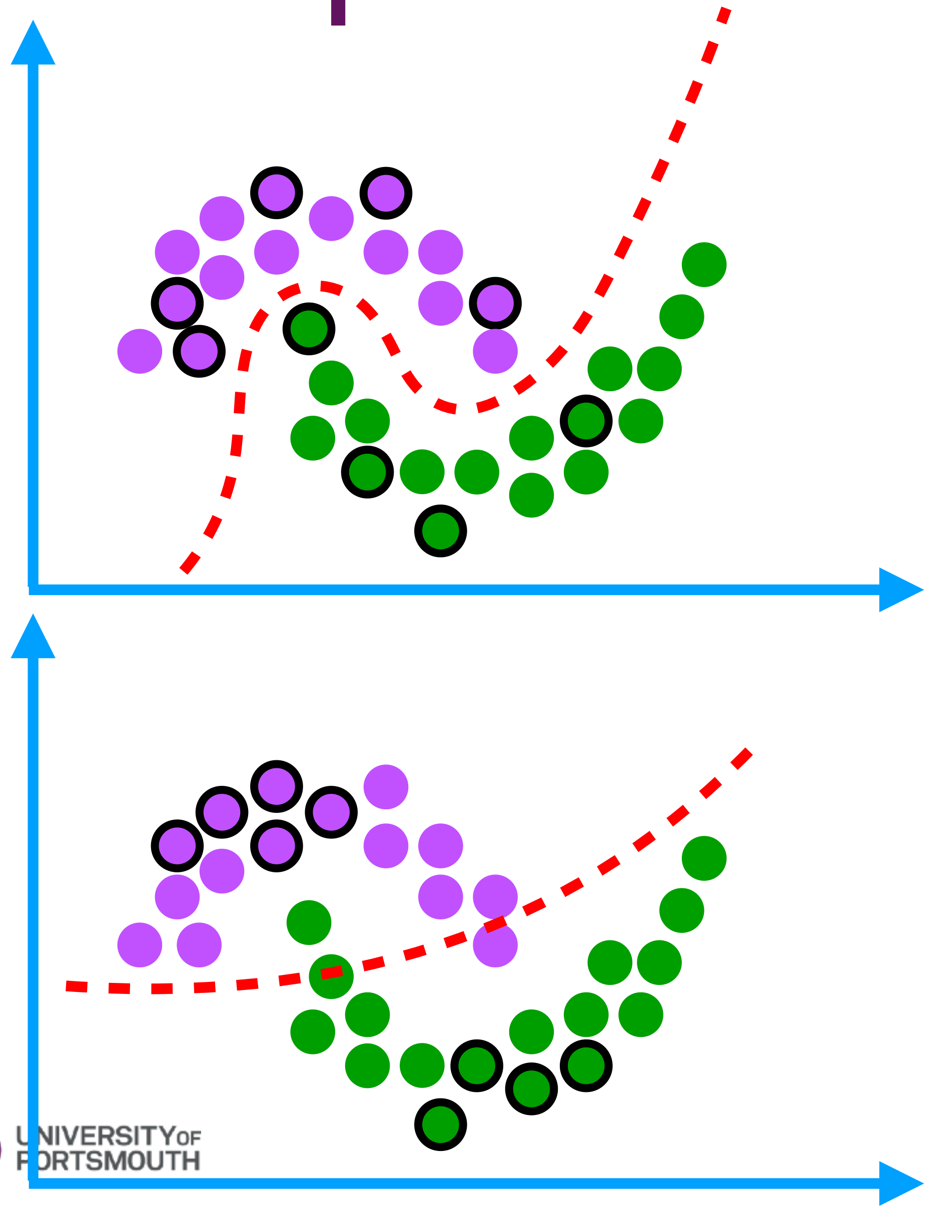
- Two 'moons' datasets (purple and green circles)
- No data are labelled
- The algorithm will first initialise a random guess and test against the inputs themselves
- There are multiple different clustering approaches and flexibility in classes. Some might do better than others…



Guess → Compare with input → adjust guess → repeat until metric reached

UNIVERSITY OF PORTSMOUTH

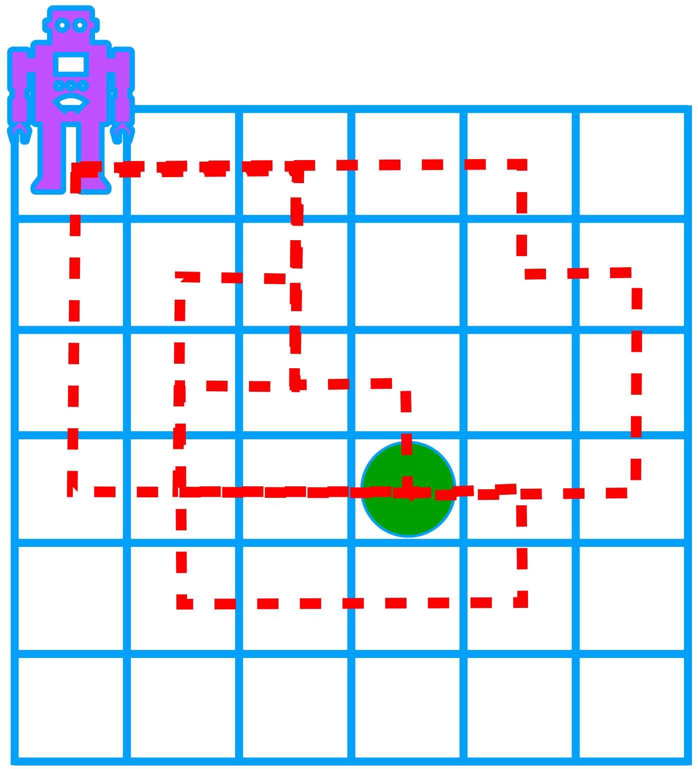
Semi-Supervised



Example with binary classification

- Two 'moons' datasets (purple and green circles)
- Some datasets are labelled (black circles)
- The labelled and unlabelled data are treated differently
- Result will depend on distribution of labelled data and algorithms chosen for clustering

Reinforcement



Example with route finding

- Robot on a grid is trying to optimise a route to the green circle
- No need to label the moves - the reward is only when the robot find the circle
- Once the robot has found it once why would it look for other routes?
- Trade off between exploration and exploitation

The Maths Behind ML

- The key maths
- How does it all work?
- How to train your dragon

- What do you need to know for this course?

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

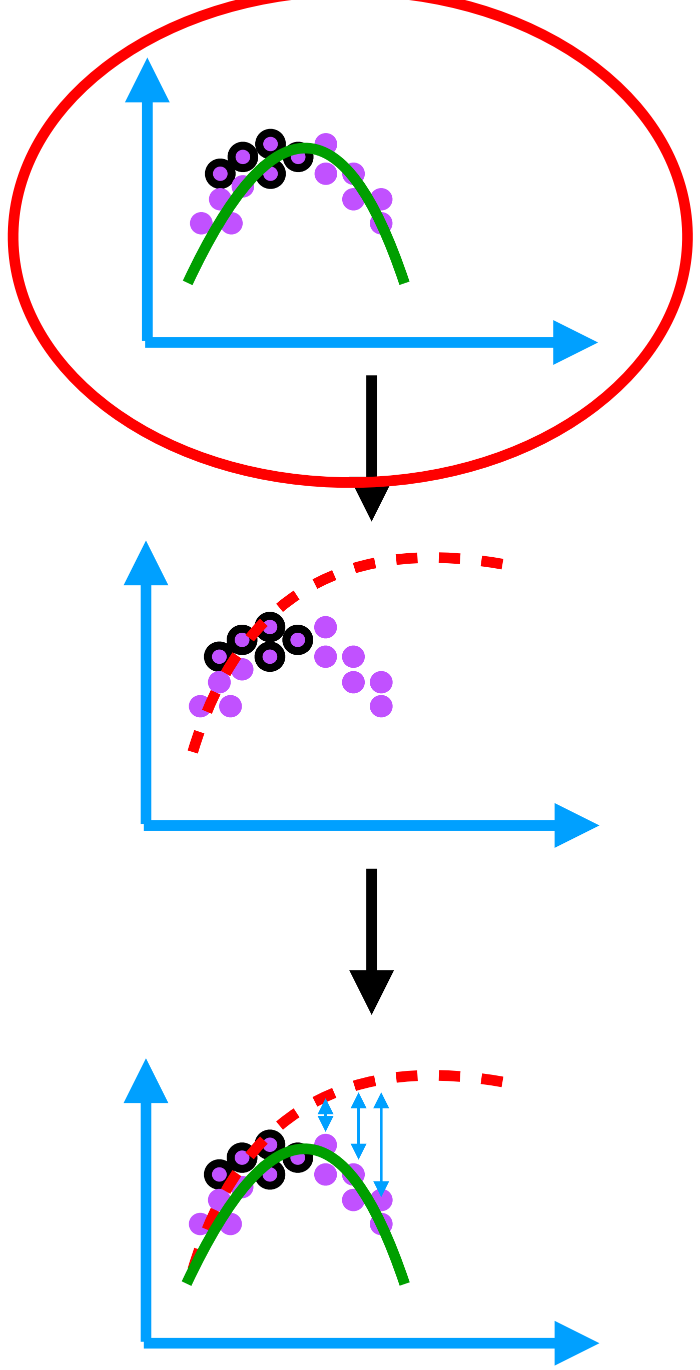
The key maths

- **Vectors and matrices** - these represent the **inputs, outputs, hidden layers** and how to transform the **parameters** which control the **features** that the algorithm optimises e.g. **weights and biases**. Some traditional ML techniques use distances between vectors or their projection onto lower dimensional planes.
- **Calculus** - this is required to optimise the network. We find derivatives in order to know if we are sitting in a minima or not and to drive the direction of optimisation of the weights, this is called '**gradient descent**'. We use the chain rule to update these derivatives through the network, this is called '**backpropagation**'.
- **Probability** - probability distributions are used often in '**cost**' functions e.g. understanding how far away from an optimal solution we are; in '**activation functions**', that is functions which allow non-linearities; and all probabilistic ML algorithms.

UNIVERSITY OF PORTSMOUTH

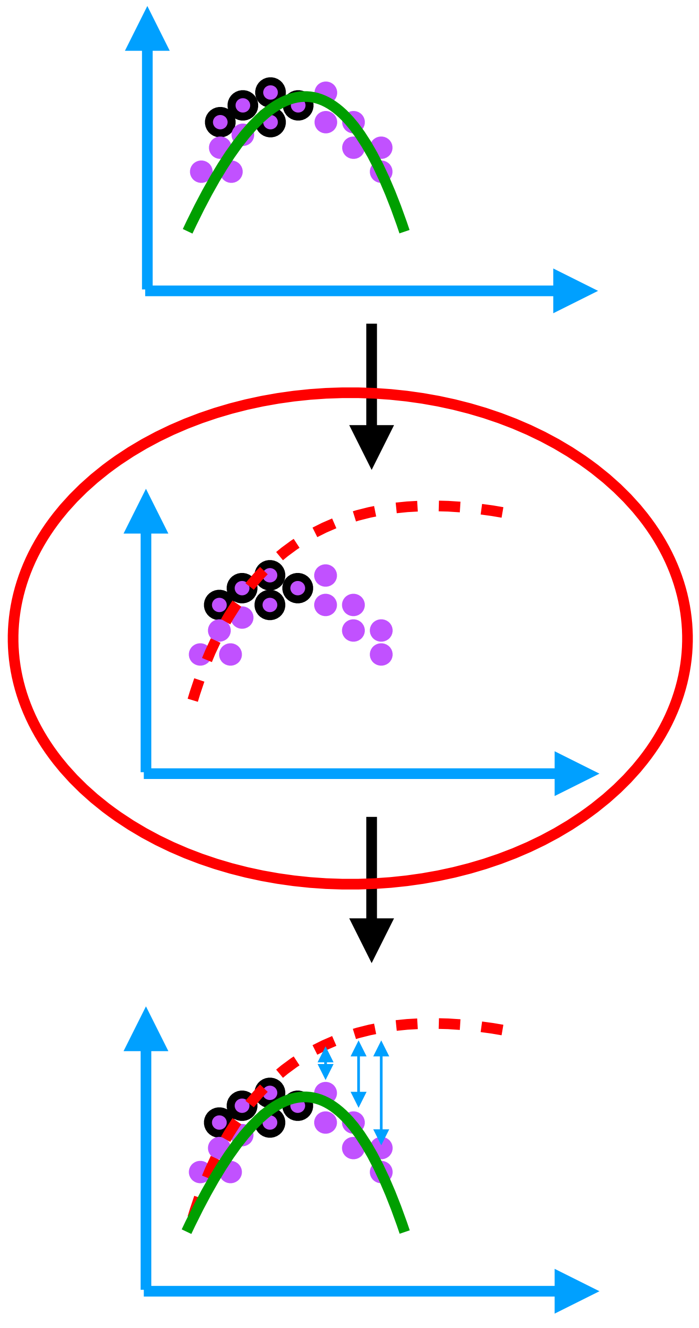
How does it all work?

- Simple supervised example:
  - Input data with some labelled 'truth'
  - Set up some algorithm with variable parameters
  - Machine takes random guess at parameters
  - Then to see how well it has done it compares the predicted value to the 'true' value (derived from labels or from a distribution of the data).



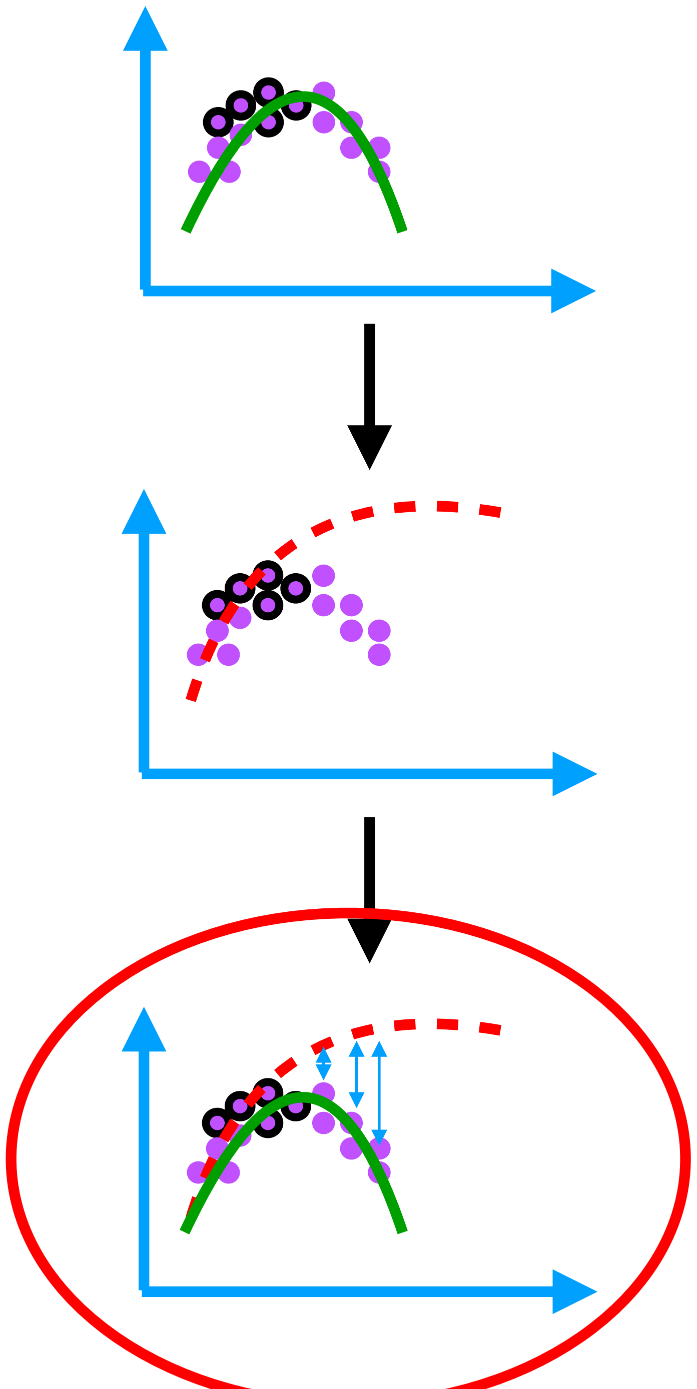
How does it all work?

- Simple supervised example:
  - Input data with some labelled 'truth'
  - Set up some algorithm with variable parameters
  - Machine takes random guess at parameters
  - Then to see how well it has done it compares the predicted value to the 'true' value (derived from labels or from a distribution of the data).



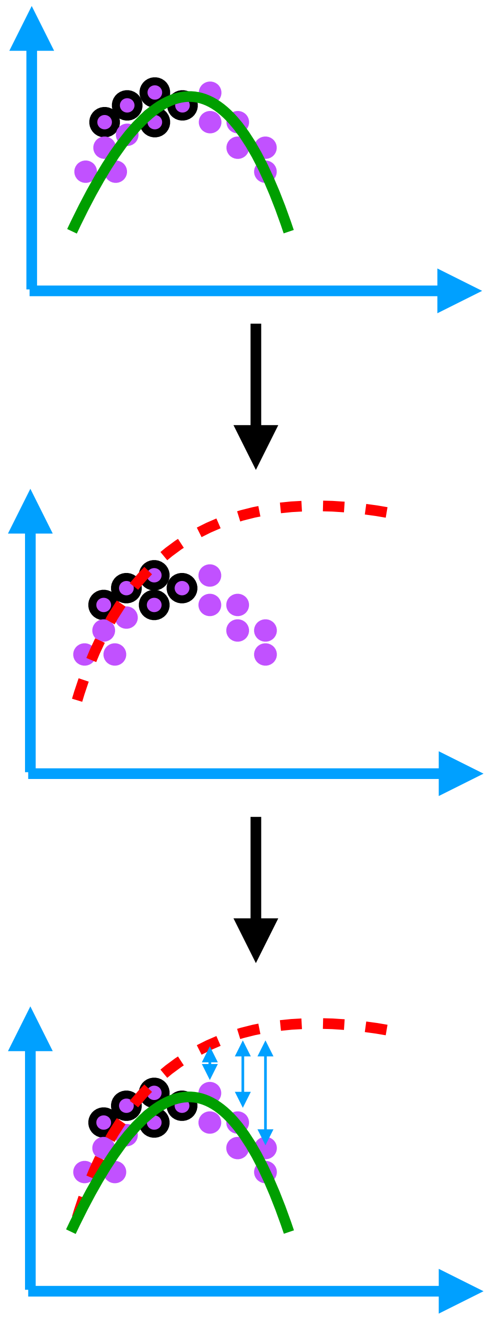
How does it all work?

- Simple supervised example:
    - Input data with some labelled 'truth'
    - Set up some algorithm with variable parameters
    - Machine takes random guess at parameters
    - Then to see how well it has done it compares the predicted value to the 'true' value (derived from labels or from a distribution of the data).
- Repeat!



How does it all work?

- Simple supervised example:
  - **Input** data with some labelled '**truth**'
  - Set up **some algorithm** with variable **parameters**
  - Machine takes random guess at parameters
  - Then to see **how well it has done** it compares the predicted value to the 'true' value (derived from labels or from a distribution of the data).
- **Repeat**



UNIVERSITY OF PORTSMOUTH

How does it all work?

- Simple supervised example:
    - **Input** data with some labelled '**truth**'

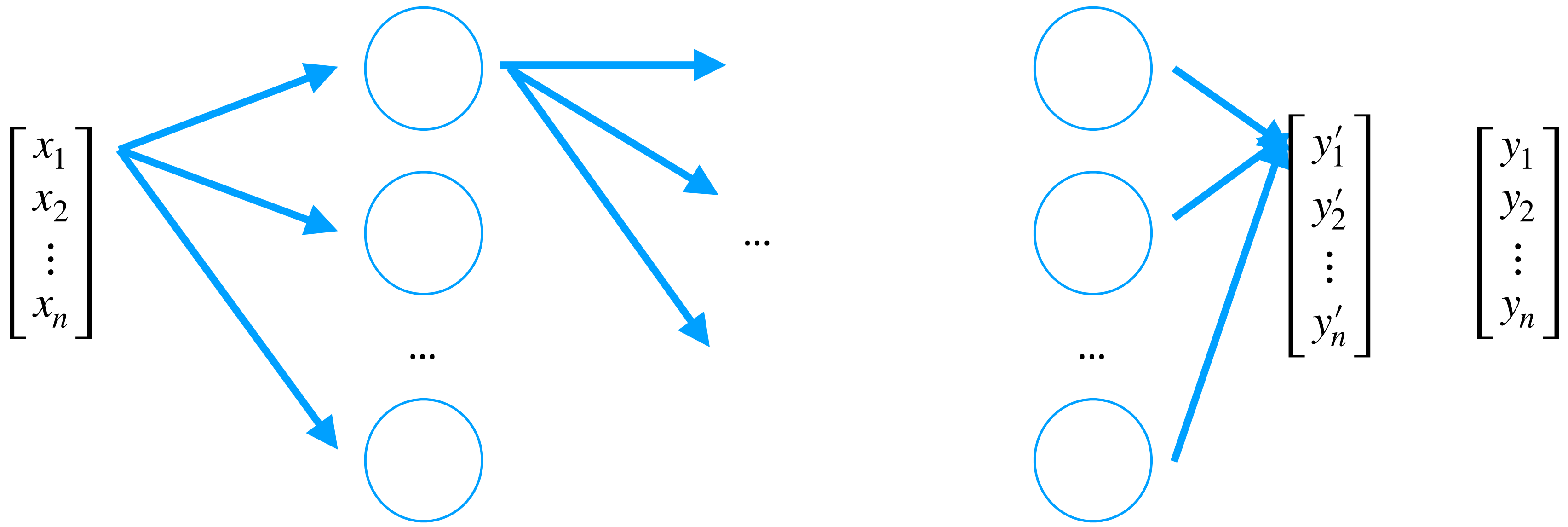
$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \begin{bmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{bmatrix} \quad \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

output

How does it all work?

- Simple supervised example:
  - Set up **some algorithm** with variable **parameters**

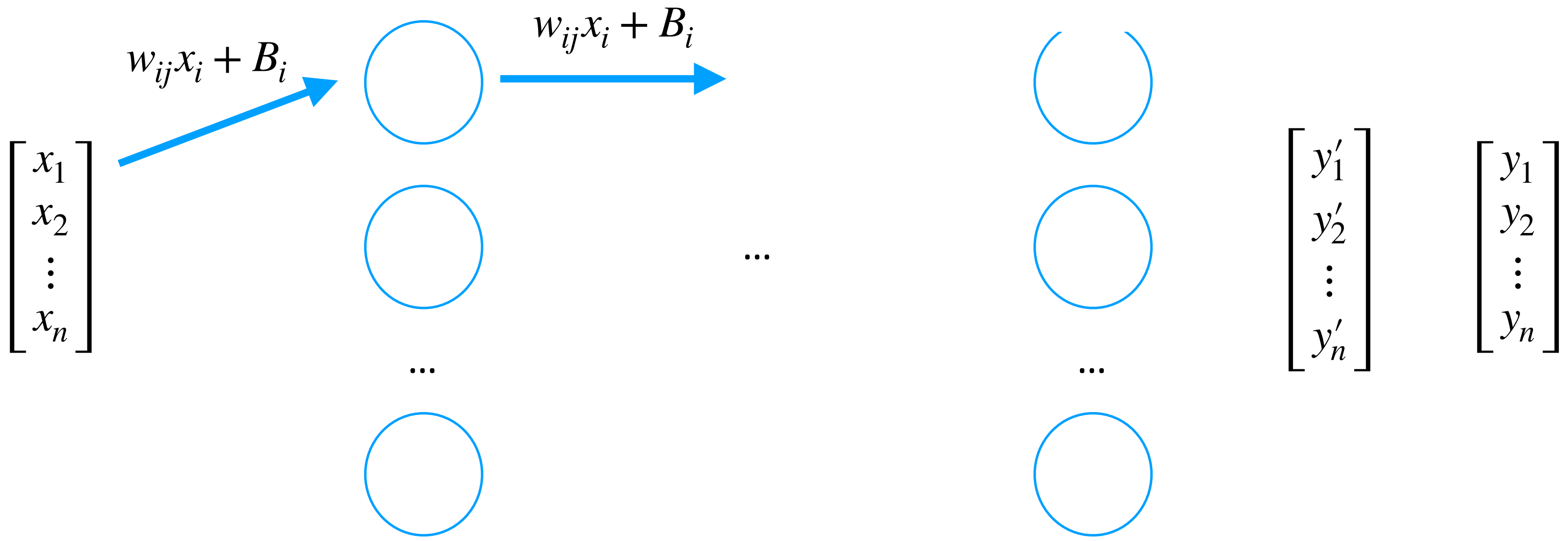
What will your network look like?
What maths will be used to connect it?



$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$
...
...
...
$\begin{bmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{bmatrix}$
$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

How does it all work?

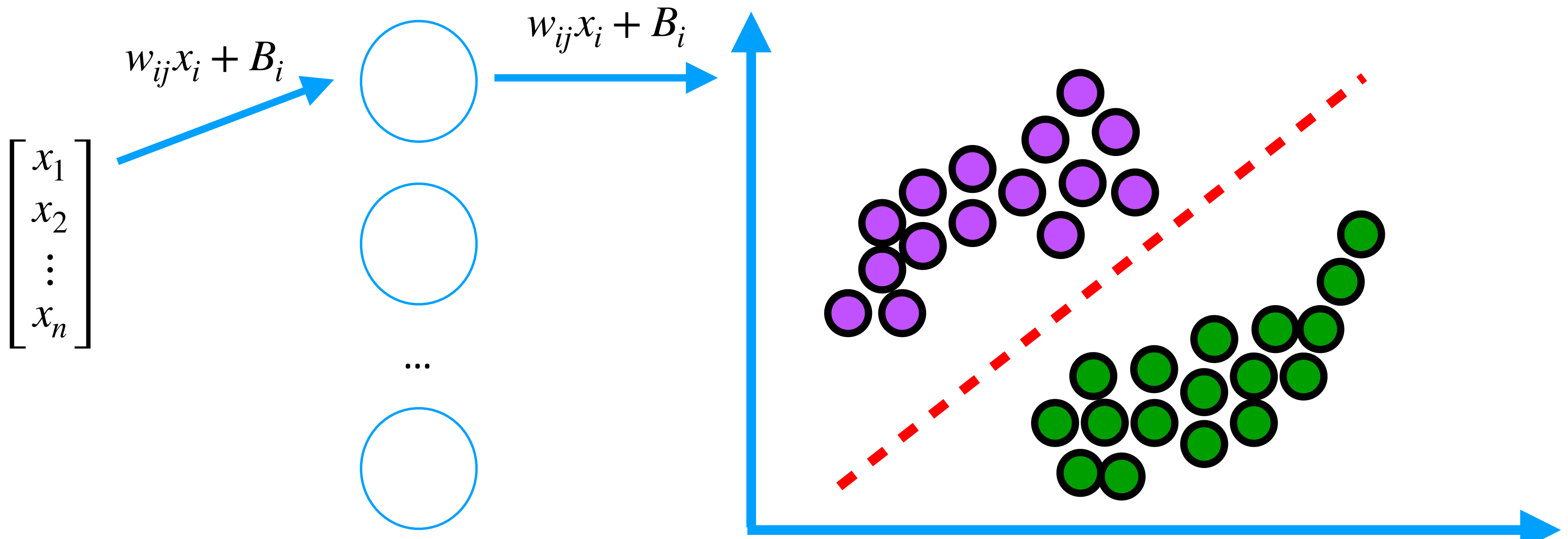
- Simple supervised example:
  - Set up **some algorithm** with variable **parameters** Weights: w_{ij} , Bias: B_i ,



$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$
$w_{ij}x_i + B_i$
$w_{ij}x_i + B_i$
...
...
...
$\begin{bmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{bmatrix}$
$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

How does it all work?

- Simple supervised example:
  - Set up **some algorithm** with variable **parameters** Weights: w_{ij} , Bias: B_i ,

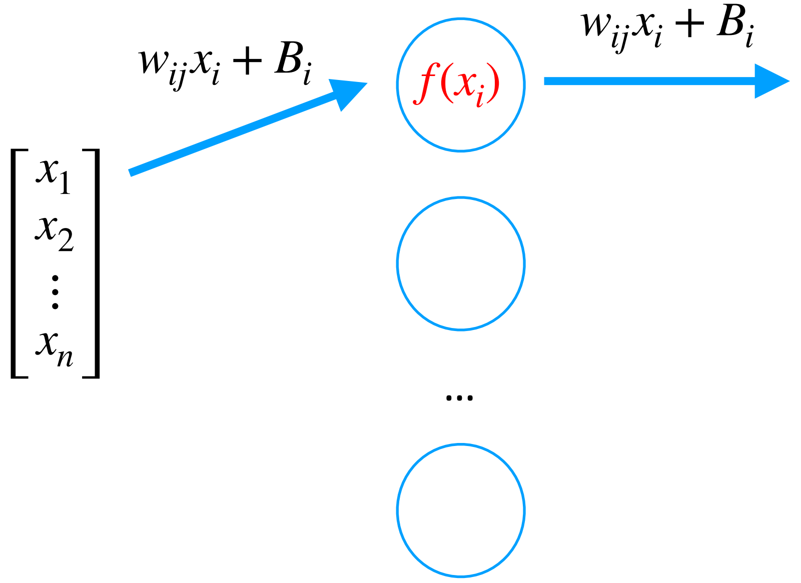


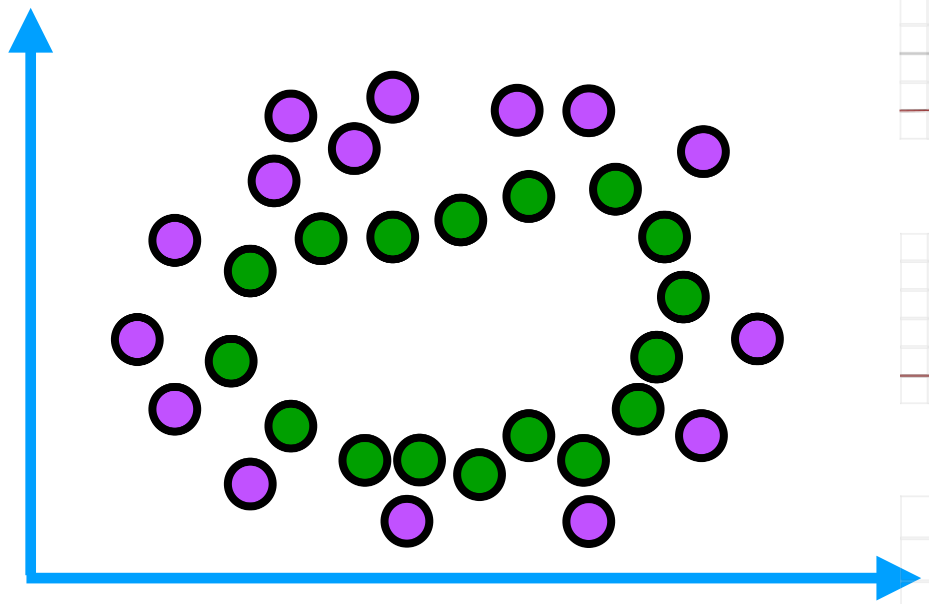
Linear (straight line) problems

How does it all work?

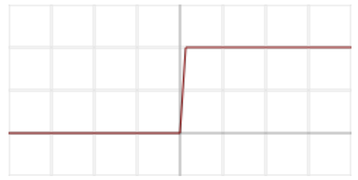
- Simple supervised example:
  - Set up **some algorithm** with variable **parameters**

Weights: w_{ij} , Bias: B_i ,
Activation function $f(x_i)$

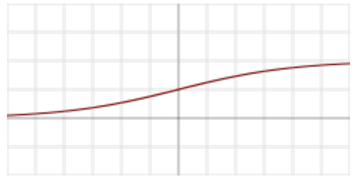




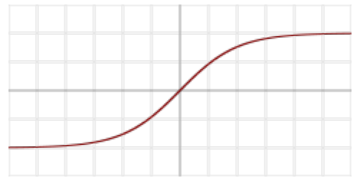
Non-linear problem?



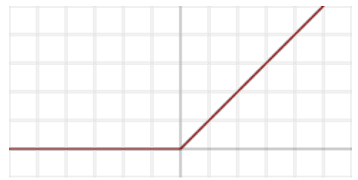
Binary step



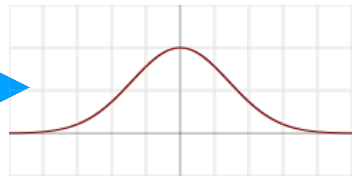
Soft max (or sigmoid)



Tanh



ReLU

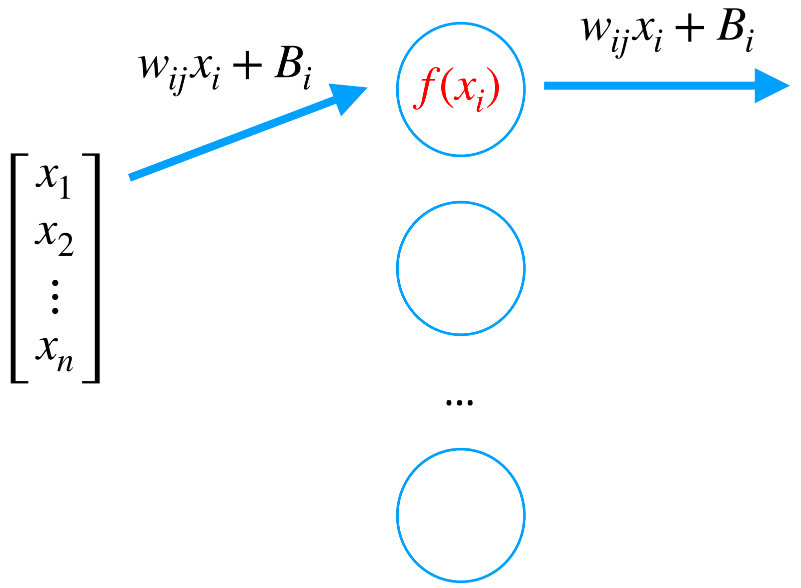


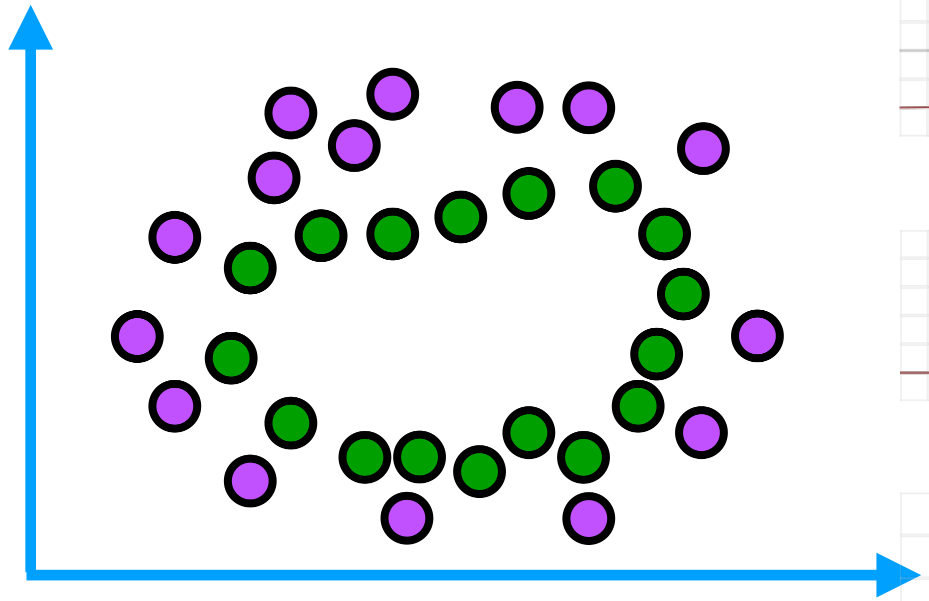
Gaussian

How does it all work?

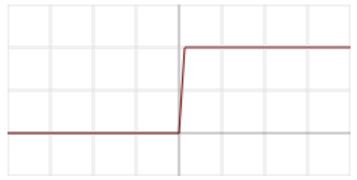
- Simple supervised example:
  - Set up **some algorithm** with variable **parameters**

Weights: w_{ij} , Bias: B_i ,
Activation function $f(x_i)$

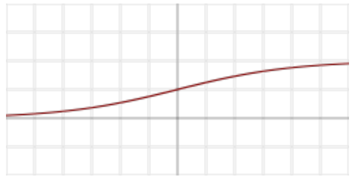




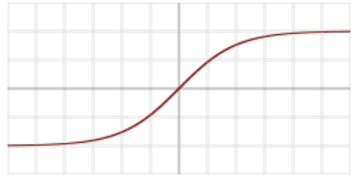
Non-linear problem?



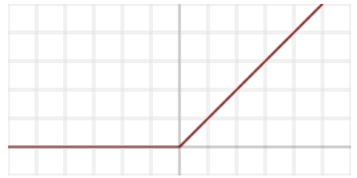
Binary step



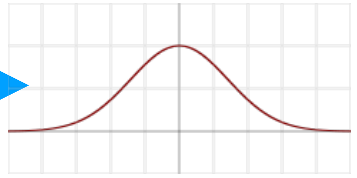
Soft max (or sigmoid)



Tanh



ReLU



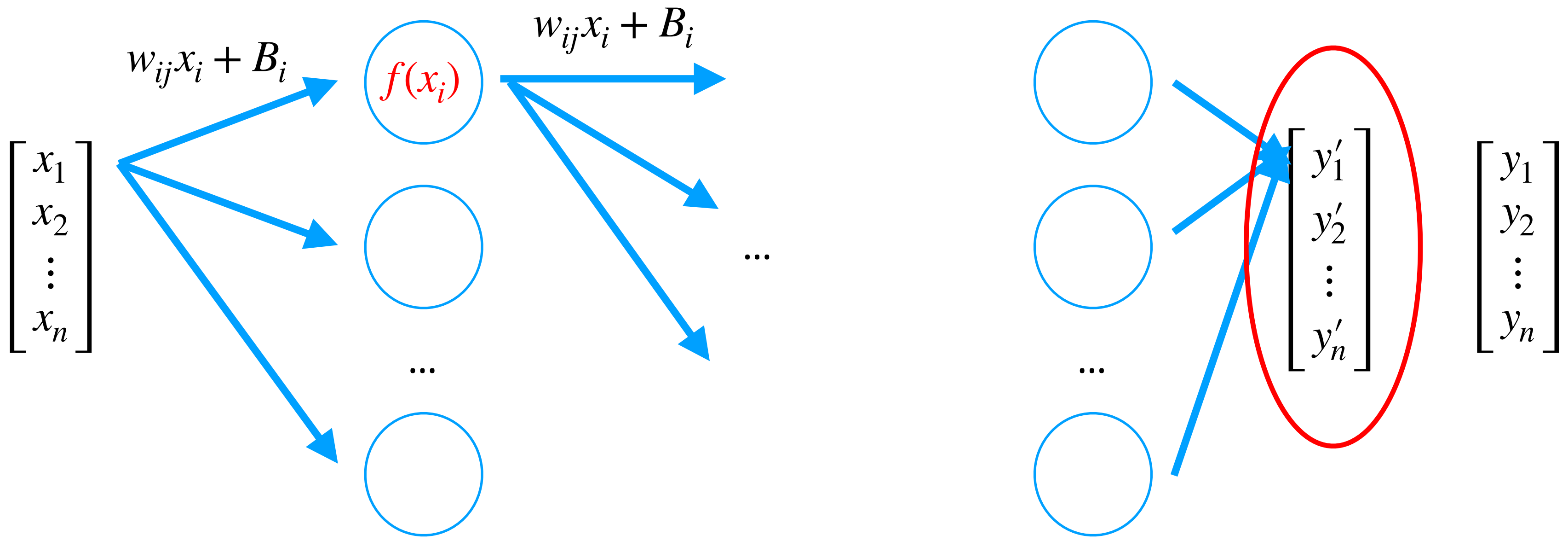
Gaussian

UNIVERSITY OF PORTSMOUTH

Hyperparameters - control learning process (not model)

How does it all work?

- Simple supervised example:
  - Machine takes random guess at parameters - generate our first outputs



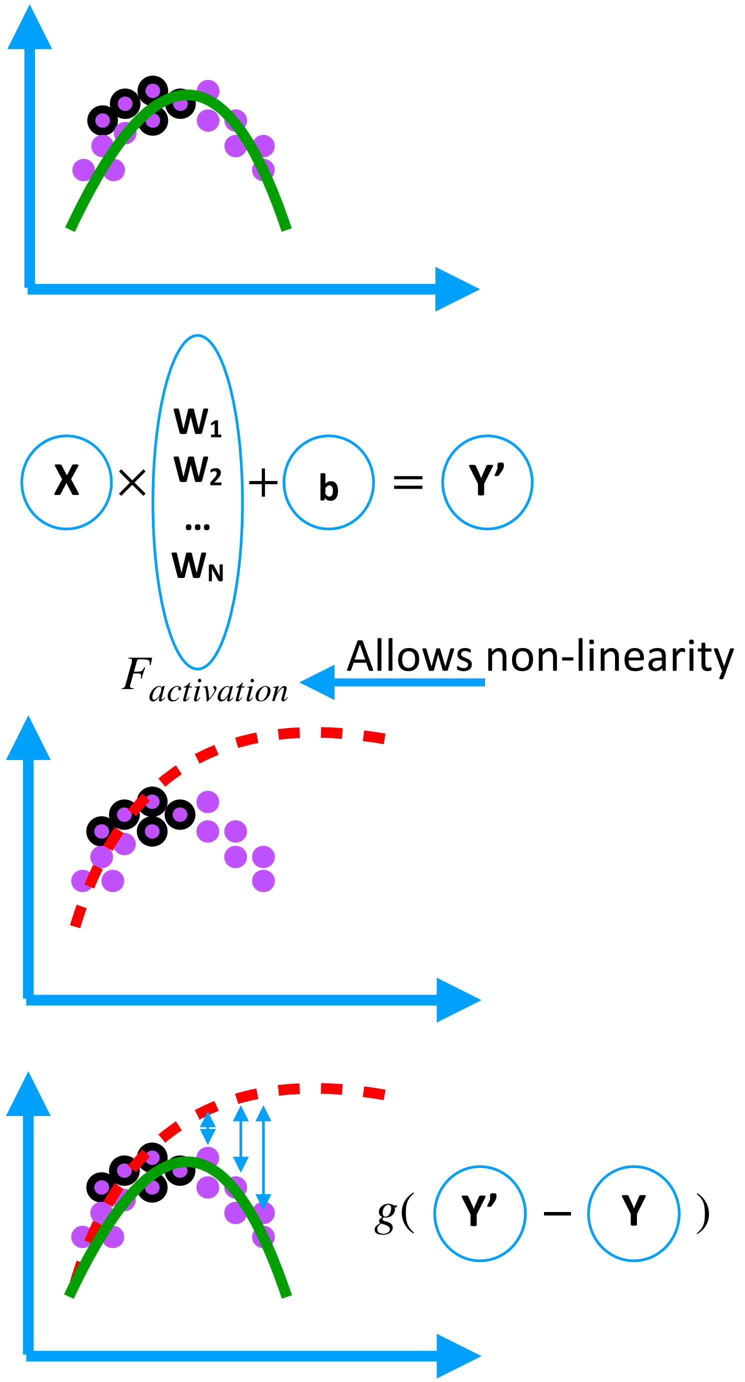
$w_{ij}x_i + B_i$
$f(x_i)$
$w_{ij}x_i + B_i$
...
...
...
$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$
$\begin{bmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{bmatrix}$
$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

How does it all work?

- To see how well it has done it compares the predicted value to the 'true' value (derived from labels or from a distribution of the data).
- It compares them using some function of the differences - '**loss**' functions or '**cost**' function or sometimes '**Error**'

$$E = g(\boxed{Y'} - \boxed{Y})$$

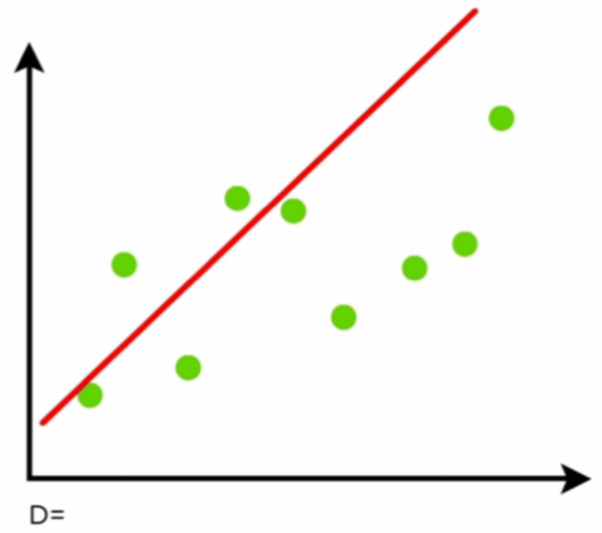
- Then it adjusts the weights and biases to try to minimise this 'loss'



Loss functions in more detail

Linear Regression

- A simple linear regression is a $y = mx + b$
- How do we evaluate our performance
- Several metrics:
  - Coefficient of determination (R^2)
  - Mean Squared Error
  - Root Mean Squared Error



R^2

$$R^2 = 1 - \frac{\sum(y_{\text{actual}} - y_{\text{predicted}})^2}{\sum(y_{\text{actual}} - \bar{y})^2}$$

- Y_{actual} are the observed data
- $Y_{\text{predicted}}$ comes from the model

- The denominator represents the variance in the data
- The numerator represents the unexplained variance after applying the regression model
- How to interpret the R2:
    - $R^2 = 1$ perfect explanation of the variance
    - $R^2 = 0$ model explains none of the variance (you may as well use the mean)
    - $R^2 < 0$ model is worse than the mean of data

UNIVERSITY OF PORTSMOUTH

R²

$$R^2 = 1 - \frac{\sum(y_{\text{actual}} - y_{\text{predicted}})^2}{\sum(y_{\text{actual}} - \bar{y})^2}$$

- How to interpret the R^2:
  - $R^2 = 1$ perfect explanation of the variance
  - $R^2 = 0$ model explains none of the variance (you may as well use the mean)
  - $R^2 < 0$ model is worse than the mean of data
- Higher R^2 not always better - you might overfit the data
- Adding more features/parameters will inflate R^2 but not mean model is better.

Mean Squared Error and Root Mean Squared Error

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_{\text{actual},i} - y_{\text{predicted},i}\right)^2$$

- MSE is the average squared different between he observed and model predictions
  - Since the difference is squared, large errors/offsets have more influence, making the MSE sensitive to outliers
- RMSE addresses this by square rooting the MSE

- Note: You might also see the MSE referred to as L2 Loss or quadratic loss

Mean Absolute Error

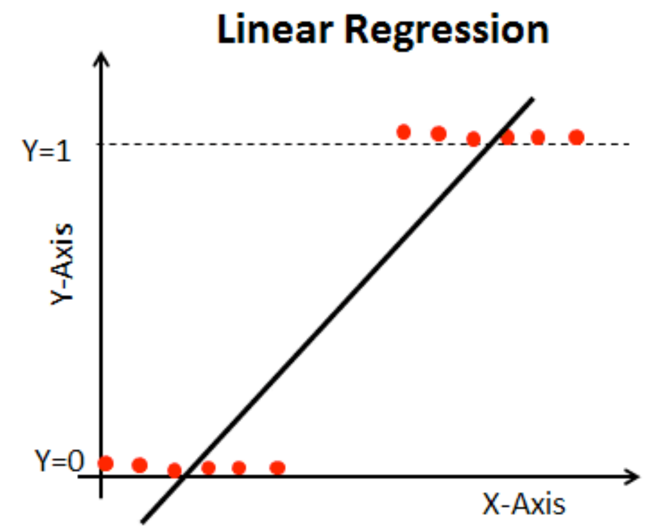
$$\frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

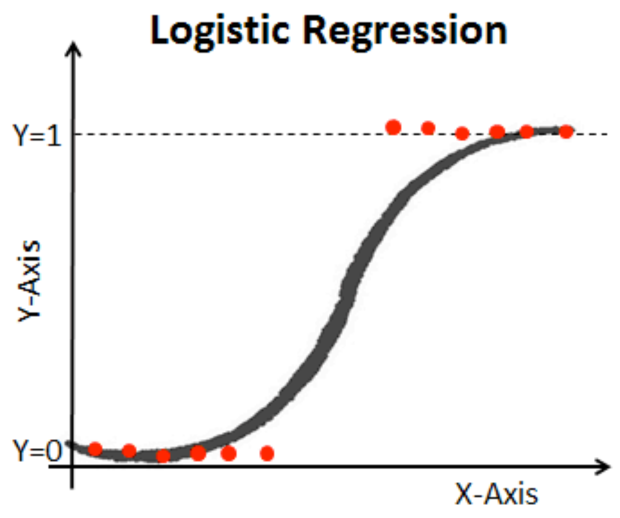
- MAE measures the absolute difference between observed and predicted
- Because it doesn't square the differences/loss, it is more robust to outliers than MSE
  - MAE is better when you don't want extreme values to influence your model too much

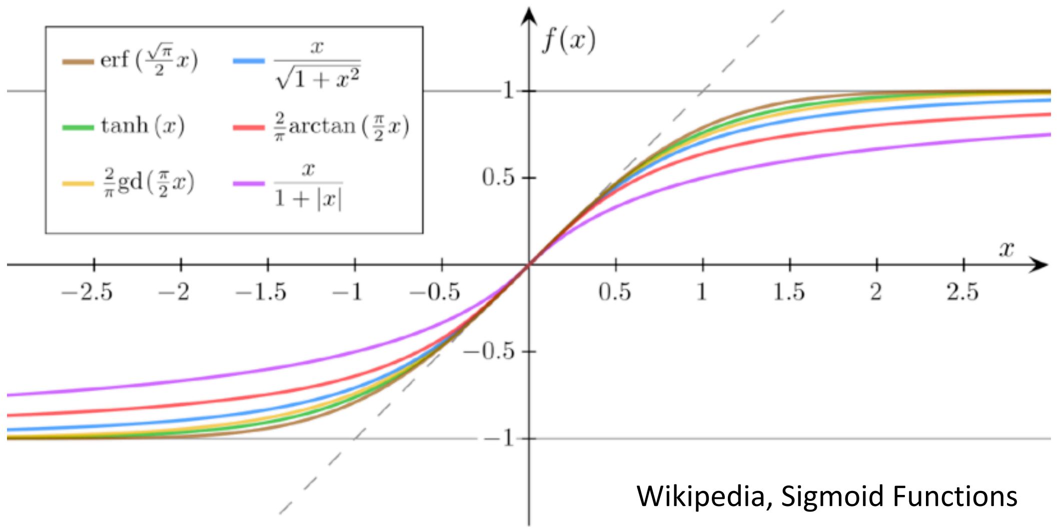
- Note: MAE is also referred to as the L1 loss

Logistic Regression

- Logistic is a special type of linear regression
- Useful for categorical, discrete, Yes/No, 1 or 0, type problems
- Predicts whether a binary event will happen



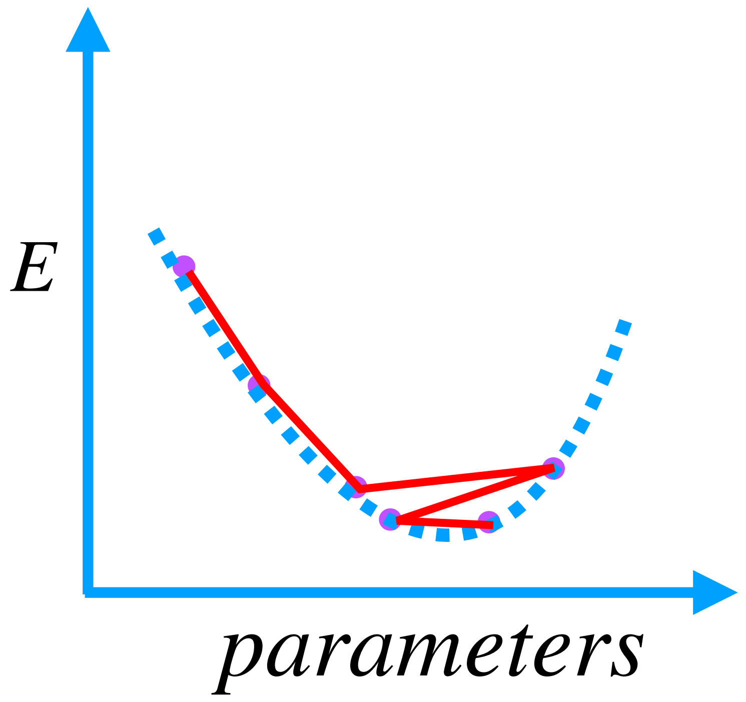




Wikipedia, Sigmoid Functions

How does it all work?

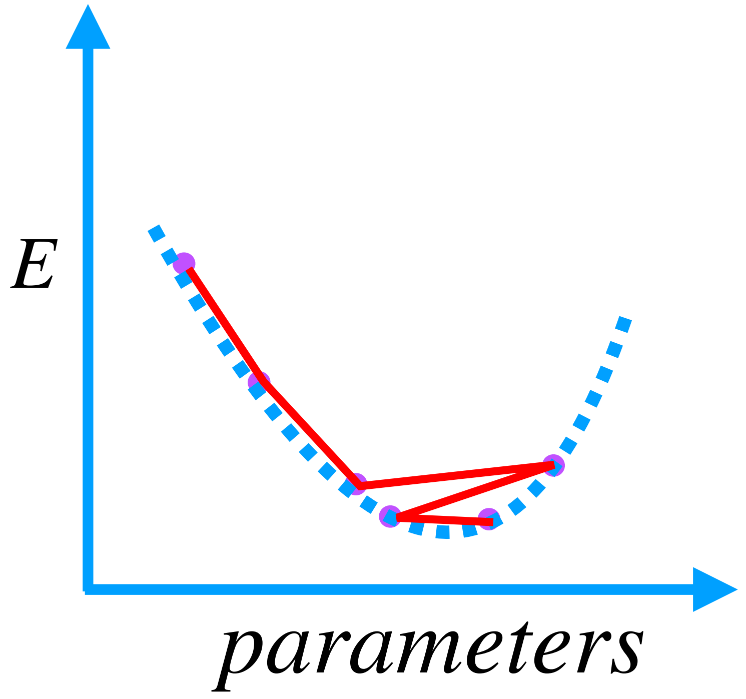
- Machine calculates new cost
- If gradient of loss is negative, i.e. changing a parameter decreased the loss and got us closer to the goal then it continues to change in this direction.
- If gradient of loss is positive i.e. we are moving away from our goal then it is less likely to step in this direction
- It will continue to optimise the solution using differentiation to calculate gradients of errors with respect to the parameters
- This is called 'gradient descent' - you differentiate the curve to find the gradient and try to descend down it
- **When there are lots of parameters how do we update them?**



How does it all work?

- Machine calculates new cost
- If gradient of loss is negative, i.e. changing a parameter decreased the loss and got us closer to the goal then it continues to change in this direction.
- If gradient of loss is positive i.e. we are moving away from our goal then it is less likely to step in this direction
- It will continue to optimise the solution using differentiation to calculate gradients of errors with respect to the parameters
- This is called '**gradient descent**' - you differentiate the curve to find the gradient and try to descend down it
- **When there are lots of parameters how do we update them?**
  - **Use the chain rule and work backward through the network taking partial derivatives of each parameter as you go**
    - **Backpropagation**

$$\frac{\partial E}{\partial a} = \frac{\partial E}{\partial b}\frac{\partial b}{\partial a}$$



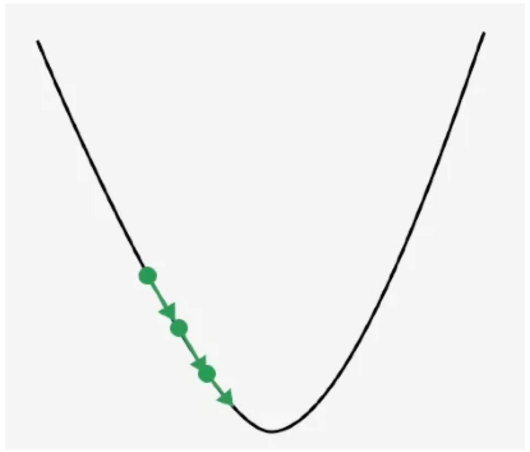
Gradient Descent in more detail

- An algorithm that estimates where a function outputs the lowest values.
- Pseudo-code (inspired by hill walking)

```
1. Dropped randomly in this scene ->
2. Calculate the gradient in all direction
   (take the partial derivative)
3. Take a step in the direction it is
   steepest
4. Repeat from step 2 until you are in a
   valley
```

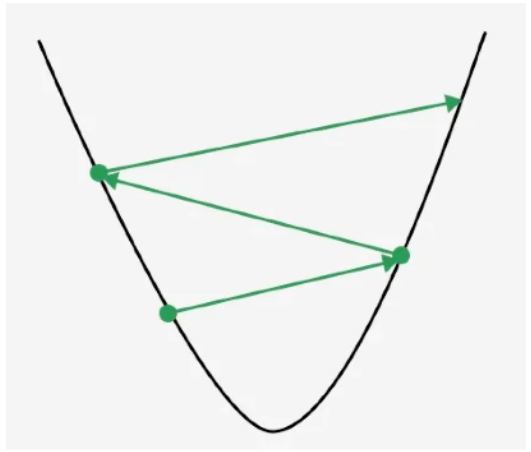
Learning Rate == Step Size

- Learning rate is a hyper parameter that is analogous to the step-size in hill walking



Small learning rate, takes a long time to converge

Large learning rate, overshoot the minimum, may never converge



Some mathematics

- Let's sick with a linear problem (the easiest!)
- Let's use the Mean Squared Error as our Loss/Cost function
- $L(m, b) = \frac{1}{n} \sum_{i=1}^{n} (y_{\text{model}} - y)^2$
- m = weight
- b = bias
- N = number of data points

Gradient of the loss function

- We now need to compute the gradient of the loss function
- $\frac{\partial}{\partial m}L(m,b) = \frac{\partial}{\partial m}\left[\frac{1}{n}\sum_{i=1}^{n}(y_{\text{model}} - y)^2\right]$ and $\frac{\partial}{\partial b}L(m,b) = \frac{\partial}{\partial b}\left[\frac{1}{n}\sum_{i=1}^{n}(y_{\text{model}} - y)^2\right]$
- $\frac{\partial}{\partial m}L(m,b) = \frac{2}{n}\sum(y_{\text{model}} - y)^2 \cdot x$
- $\frac{\partial}{\partial b}L(m,b) = \frac{2}{n}\sum(y_{\text{model}} - y)^2$
- Let's look at some code…

UNIVERSITY OF PORTSMOUTH

The key maths

- **Vectors and matrices** - these represent the **inputs, outputs, hidden layers** and how to transform the **parameters** which control the features that the algorithm optimises e.g. **weights and biases**. Some traditional ML techniques use distances between vectors or their projection onto lower dimensional planes.
- **Calculus** - this is required to optimise the network. We find derivatives in order to know if we are sitting in a minima or not and to drive the direction of optimisation of the weights, this is called '**gradient descent**'. We use the chain rule to update these derivatives through the network, this is called '**backpropagation**'.
- **Probability** - probability distributions are used often in '**cost**' functions e.g. understanding how far away from an optimal solution we are; in '**activation functions**', that is functions which allow non-linearities; and all probabilistic ML algorithms.

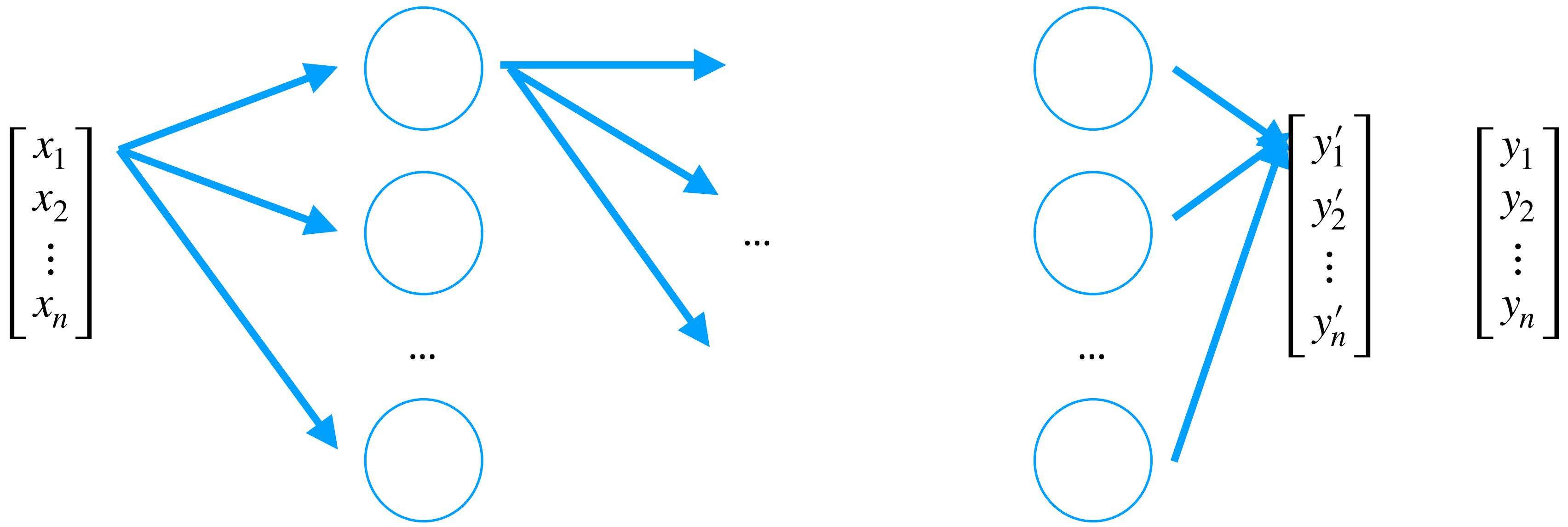
UNIVERSITY OF PORTSMOUTH

Features are any measurable property provided to the network
Feature engineering - choosing what inputs to give (e.g. colour)

Feature also used when interpreting network
Here is means what did the network decide is important
There are visualisations to interpret these BUT they can be misleading - **don't over-interpret a complicated network**

How does it all work?

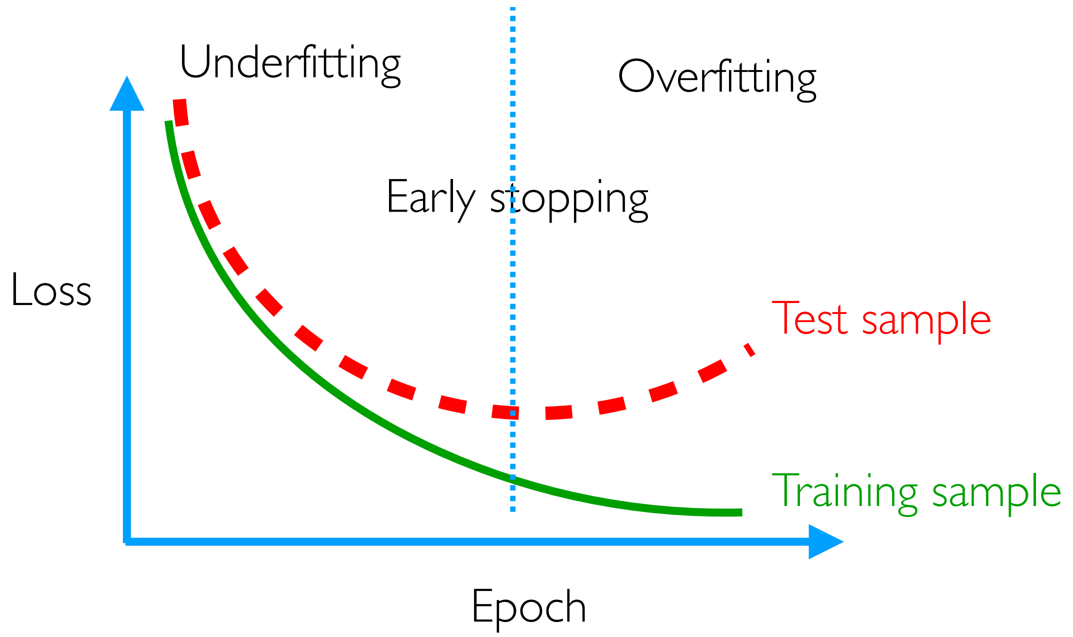
- Simple supervised example:
  - Set up **some algorithm** with variable **parameters**



$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$
...
...
...
$\begin{bmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{bmatrix}$
$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

How to train your Dragon

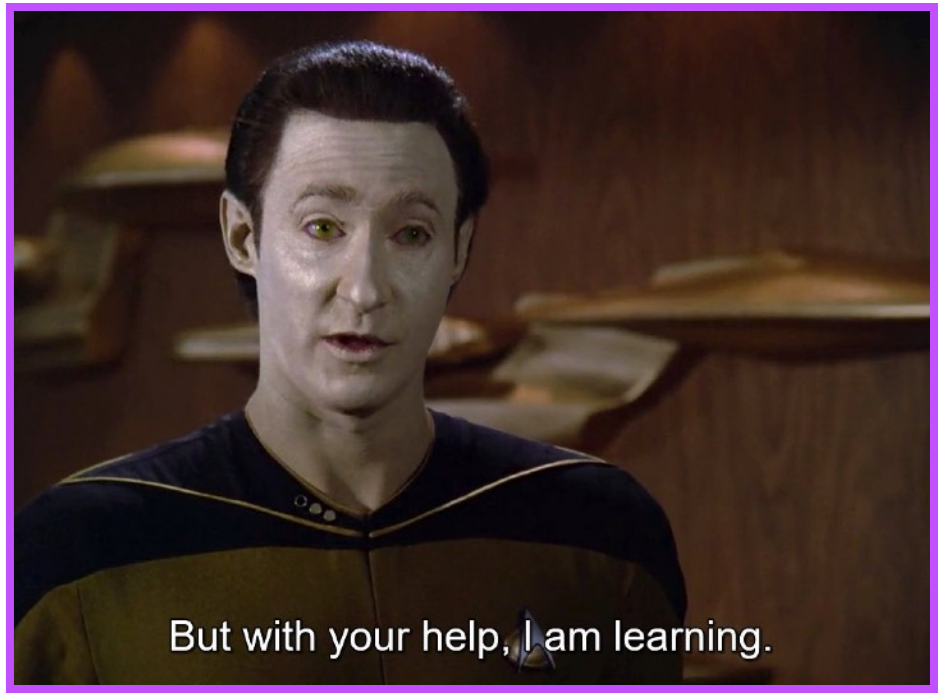
- Hyperparameters define our learning process e.g.
  - Learning rate
  - Learning iterations (e.g. batch size and number of epochs)



- There are multiple metrics one can use to define early stopping and the 'best' will depend on the situation. Keep track and monitor metrics throughout the training processes.

The Most Important Part: Data

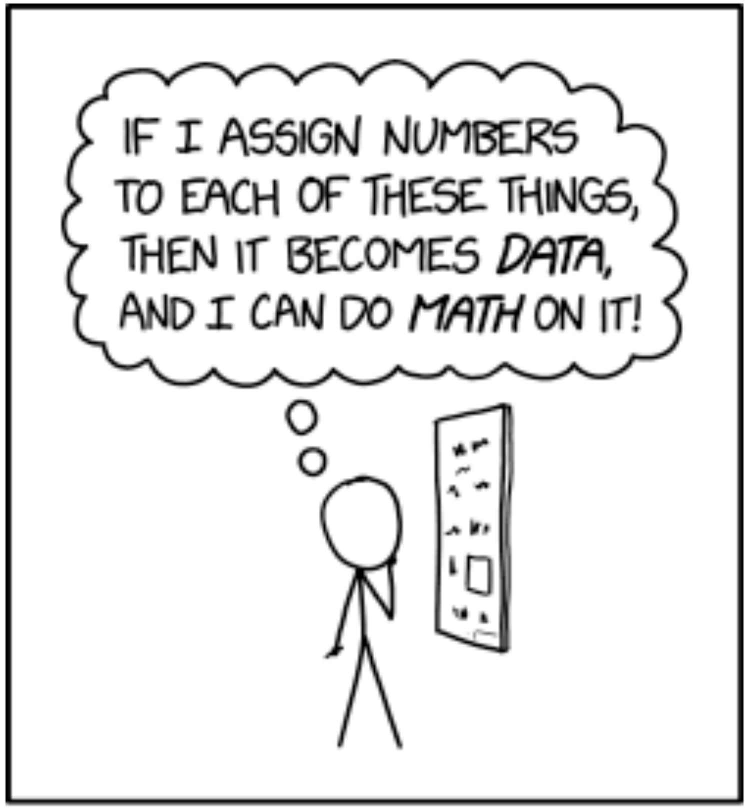
- Data collection and prep
- Feature engineering - why normalise?
- Data augmentation
-



UNIVERSITY OF PORTSMOUTH

Data collection and prep

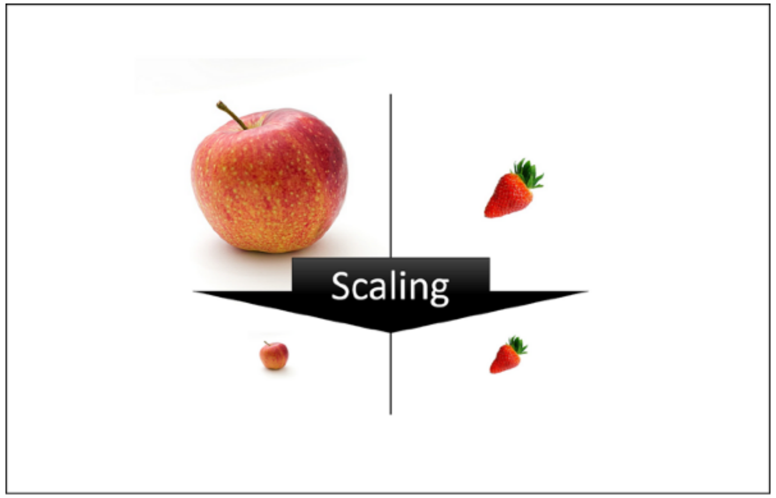
- What is your objective?
- What data encompasses this objective
- Can you source this data ethically? This encompasses both permissions for use in training and for the use case of the data.
- Is your data labelled and if not how will you label it or model it?


IF I ASSIGN NUMBERS TO EACH OF THESE THINGS, THEN IT BECOMES *DATA*, AND I CAN DO *MATH* ON IT!

THE SAME BASIC IDEA UNDERLIES GÖDEL'S INCOMPLETENESS THEOREM AND ALL BAD DATA SCIENCE.

Feature engineering

- Explore your data - choose your features!
- Feature scaling
    - Your raw data will contain vastly different scales
    - Feature scaling is the process of rescaling data in order that they are on the same scale for the machine to interpret.
    - If we do not scale correctly then the algorithm will over-emphasise arbitrarily higher weighted features.
    - At best this takes the machine longer to optimise and at worse significantly biases the results.
- Normalisation [0-1] or standardisation are the common methods

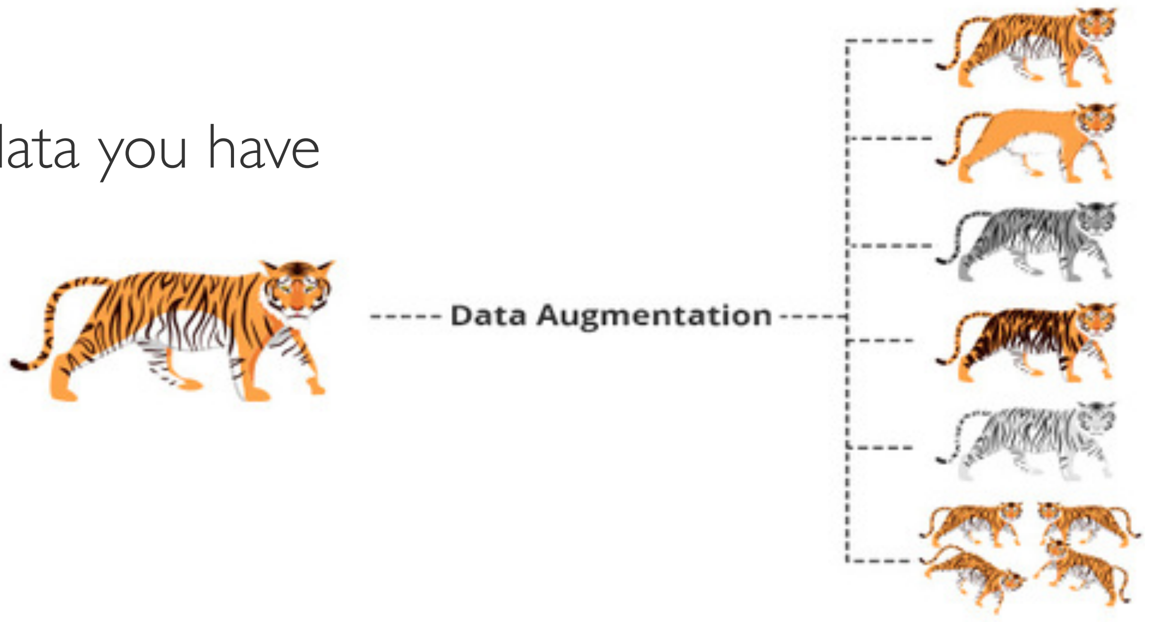


Feature engineering

- You should also be aware of:
  - Missing data - do you need to 'fill' it and how?
  - Outliers - do you remove them?
  - The distributions of your data - are they skewed? Do you want them to be more gaussian?

Data augmentation

- Creating 'new' data by augmenting the data you have
  - Translation (spatial and temporal)
  - Rotation
  - Scaling (size, volume, pitch)
  - Mirror images
  - Cropping
  - Adding noise
  - Changing backgrounds
  - Use synonyms
  - Masking



UNIVERSITY OF PORTSMOUTH

What will be used in this course?

- The Data is always the most important part
- We will use a variety of data prep techniques in scikit-learn and pytorch (and all codes have modules for these techniques) but you can easily write and code your own data augmentation routines - it depends on the data and use case

The Tools of ML

- We'll be using Python - flexible, open source, lots of modules, pre-trained modules and test datasets
- There are many codes one could use… Scikit-learn, PyTorch, Tensor Flow, Keras, Jax, …
- Use the one(s) that provide the easy access to all the tools you want for a specific problem
- Scikit-learn is great for traditional ML / AI (good ol' fashioned AI and non-deep learning techniques)
- Intro to Scikit-learn and PyTorch next week
- **Tuesday 18th November 2025, Anglesea 2.02, 4-6pm**