How Neural Networks Work

GAN: good for image generation
CNN: good for image recognition > each input neuron represents a pixel
Long short-term memory network: good for speech recognition

A plain neural network is also known as a **multilayer perceptron**

A neuron holds a number between 0 and 1
- where 0 is off (not activated) and 1 is on (activated)
- The number it holds is called an **activation**

For example, if an image is in greyscale, 0 would represent a black pixel and 1 would represent a white pixel

The output neuron is also between 0 and 1, and represented a confidence or probability, so how correct the network thinks the prediction is

There are x number of hidden layers in between the input and output layer
The activations of one layer determine the activation of the next layer
Each layer causes a specific pattern in the next layer, unique to each input, so by the end the output recognises the specific pattern and can classify/generate etc

Layers

Weights are assigned to each connection in between layers
These can be positive or negative

You would take all the **activations** connected to a neuron from the previous layer and calculate the **weighted sum** to get the activation for the neuron
- To make the activations a number 0<=x<=1, put the weighted sum into a formula for example the most common one is the **sigmoid function** (logistic curve)
- a(i) = d(Wa(0) + b) layer 1 activation = sigma(weight * activation from layer 0 + bias)

You can add **bias** to the network, for example maybe you want the weighted sum to be >5, then you would add in an extra term to the sigmoid function that would be -5
- This allows you to shift the data left or right essentially, can lead to more accurate predictions

The **activation function** will decide whether neuron will be activated or not

If fully connected network, each neuron in one layer will connect to each neuron in the next layer

When referring to ‘learning’ this is just getting the computer to learn and find the most appropriate weights and biases for each neuron
Neural networks do this learning via gradient descent which finds minimas

Cost functions help networks distinguish better
It adds up all the squares of differences between output activations and actual value so the number is small when more correct and large when there is a bad prediction

Then find the average cost of all training data, which will show how good or bad the network is

How do we find a minimum output for 1 input? **Gradients**!

- Start at a random slope and walk from there, going left or right to find the minimum
- Go left if input is positive and right if input is negative
- You will eventually find a local minimum of a function

For two inputs (x, y): find which direction decreases the function the quickest to find a minimum (like a ball rolling down a hill)

To compute a vector size and how steep it is, use gradient descent

- Add these gradients to the cost function to nudge it to the minima iteratively, minus it to get the cost closer to 0

A network learning is just minimising a **cost function**

Gradient descent = process of repeatedly nudging the input of a function by some multiple of the negative gradient to make cost function closer to zero

Size of vector weight tells you how much a weight matters