

INTRODUCTION TO AI – NEURAL NETWORKS

Philip Mortimer

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.



RECAP

- Discussed various approaches to coding a chess AI
- Approach #1 – random moves
- Discussed briefly that random moves can be cleverly used with other algorithms to produce domain independent learning
- Approach #2 – depth limited minimax using weighted sum of pieces
- #2 plays good chess though lacks positional understanding
- Discussed use of opening database and more positional AI for approach #3
- Approach #4 briefly looked at supervised learning. Discussed benefits and shortcomings of model
- Approach #5. We looked at how Stockfish (professional chess engine) used iterative deepening minimax along with neural network.

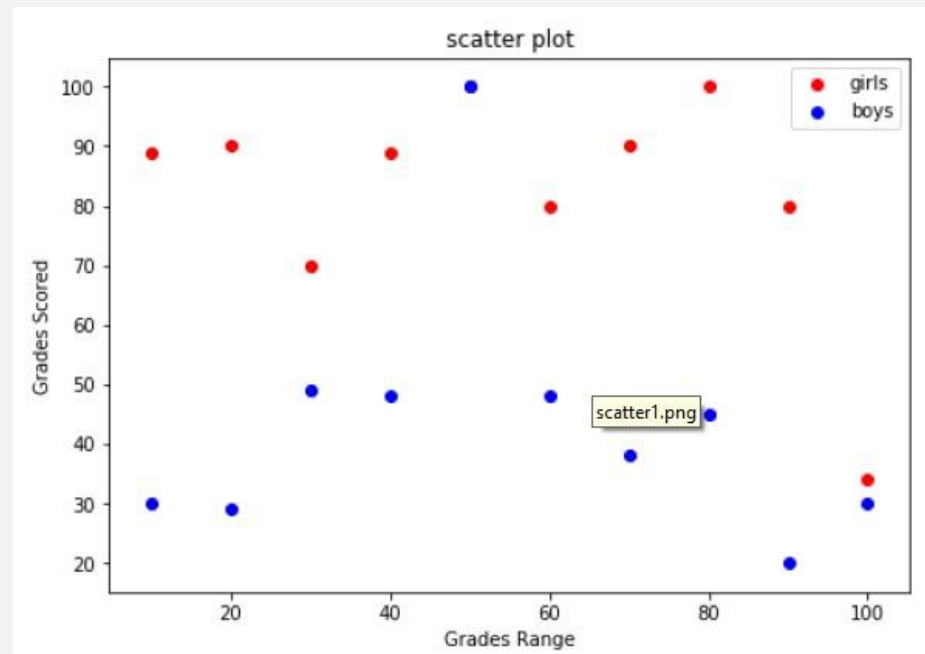
OVERVIEW

- We'll talk about Neural Networks – the structure used in lots of cutting edge ML systems.
- We'll explain supervised learning in detail
- We'll cover supervised learning techniques like stochastic gradient descent
- The topics mentioned are going to be complicated and it's completely normal to not understand everything for the first time. You should hopefully get a high level understanding and also a foundation which can be expanded through other forms of AI exposure
- We'll contextualise the theory with a practical example
- By the end of this lesson, you'll have trained your first neural network

NEURAL NETWORKS

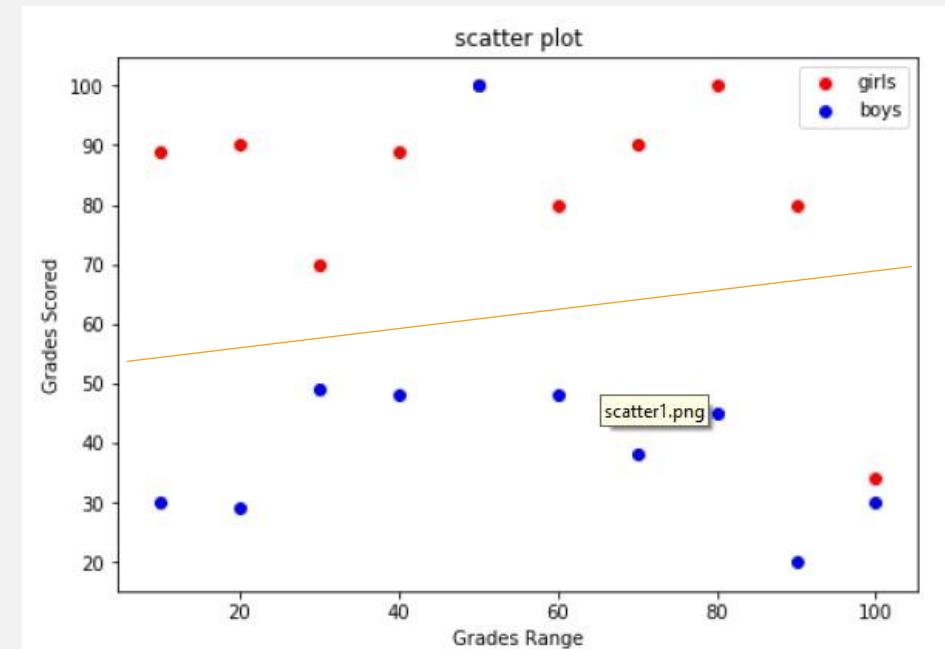
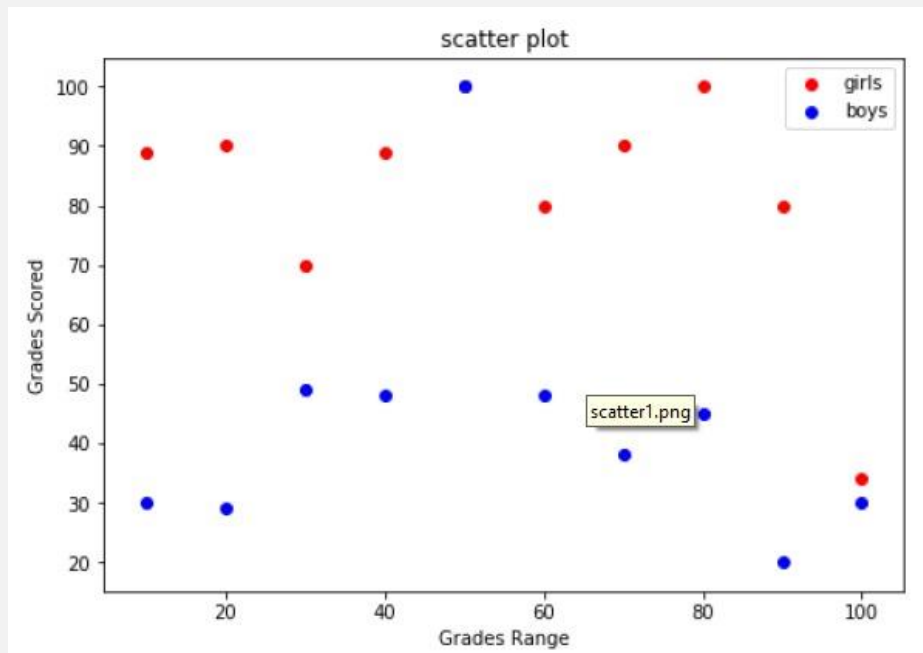
- Our brains contain billions of units called neurons. These neurons are typically modelled as either on or off. These neurons are connected together with various biological paths
- Neural networks are loosely designed to imitate this structure of the human brain
- To understand how neural networks we first need to look at a simpler problem

PERCEPTRON'S



PERCEPTRON'S

- We need to draw one line that partitions the green values and the red values.
- We would like this line to be the best line (i.e. get as many points as possible to be on the correct side of the line)
- This is a binary classification task



PERCEPTRONS

- To draw this line, we can use a technique known as linear regression
- Simply apply a mathematical formula to data (something which a computer can do really quickly)
- This can be extended to more groups by taking a weighted sum of the values

For a sample of n pairs of observations (x_i, y_i)

$$S_{xx} = \sum (x_i - \bar{x})^2 = \sum x_i^2 - \frac{(\sum x_i)^2}{n}$$

$$S_{yy} = \sum (y_i - \bar{y})^2 = \sum y_i^2 - \frac{(\sum y_i)^2}{n}$$

$$S_{xy} = \sum (x_i - \bar{x})(y_i - \bar{y}) = \sum x_i y_i - \frac{(\sum x_i)(\sum y_i)}{n}$$

A measure of linear association between two variables X and Y is given by the Pearson product - moment correlation coefficient r .

For the sample $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, it is given by $r = \frac{S_{xy}}{\sqrt{S_{xx}S_{yy}}}$.

Given data, the parameters α and β of the linear regression model may be estimated using the principle of least squares.

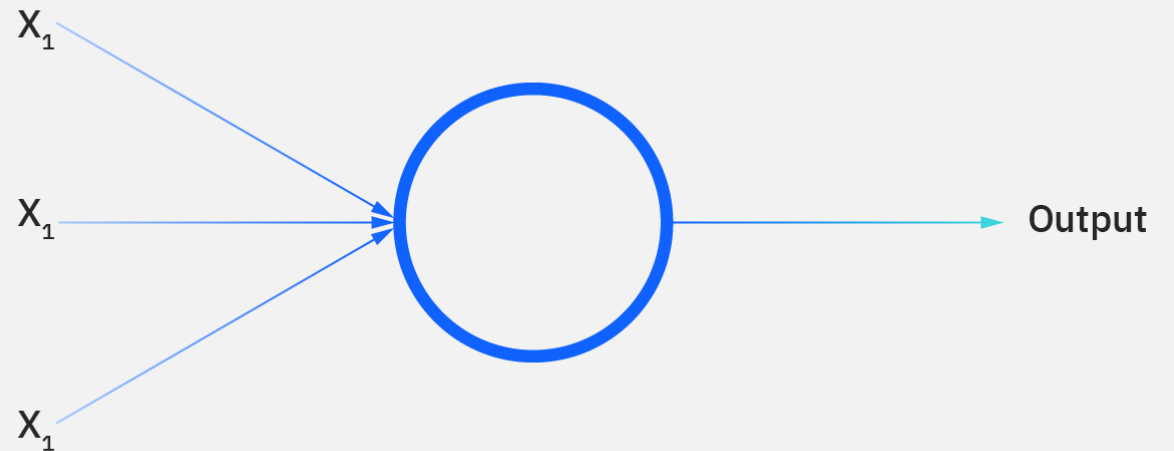
The least squares estimate $\hat{\beta}$ of the parameter β is given by $\hat{\beta} = \frac{S_{xy}}{S_{xx}}$.

The least squares estimate $\hat{\alpha}$ of the parameter α is given by $\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$.

The least squares regression line is given by $y = \hat{\alpha} + \hat{\beta}x$.

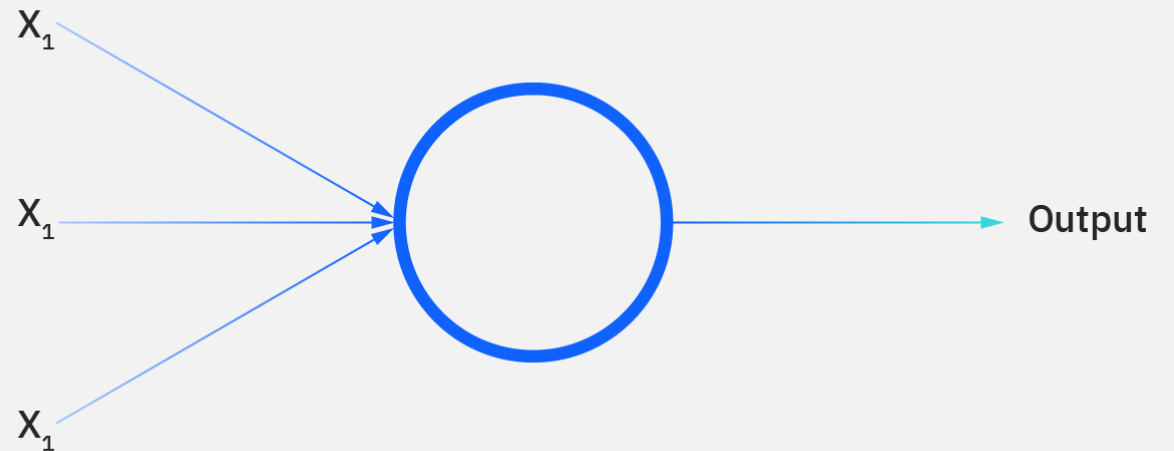
PERCEPTRON

- Perceptrons can be thought of as their own linear regression models.
- For a linear regression model, the line we needed to draw was $y = xw + b$ Where w and b are constants selected by the formula.
- This line could be extended to more than two groups by drawing an n dimensional vector: $v = (\sum_{i=0}^n x_i w_i) + b$ Where w and b are constants



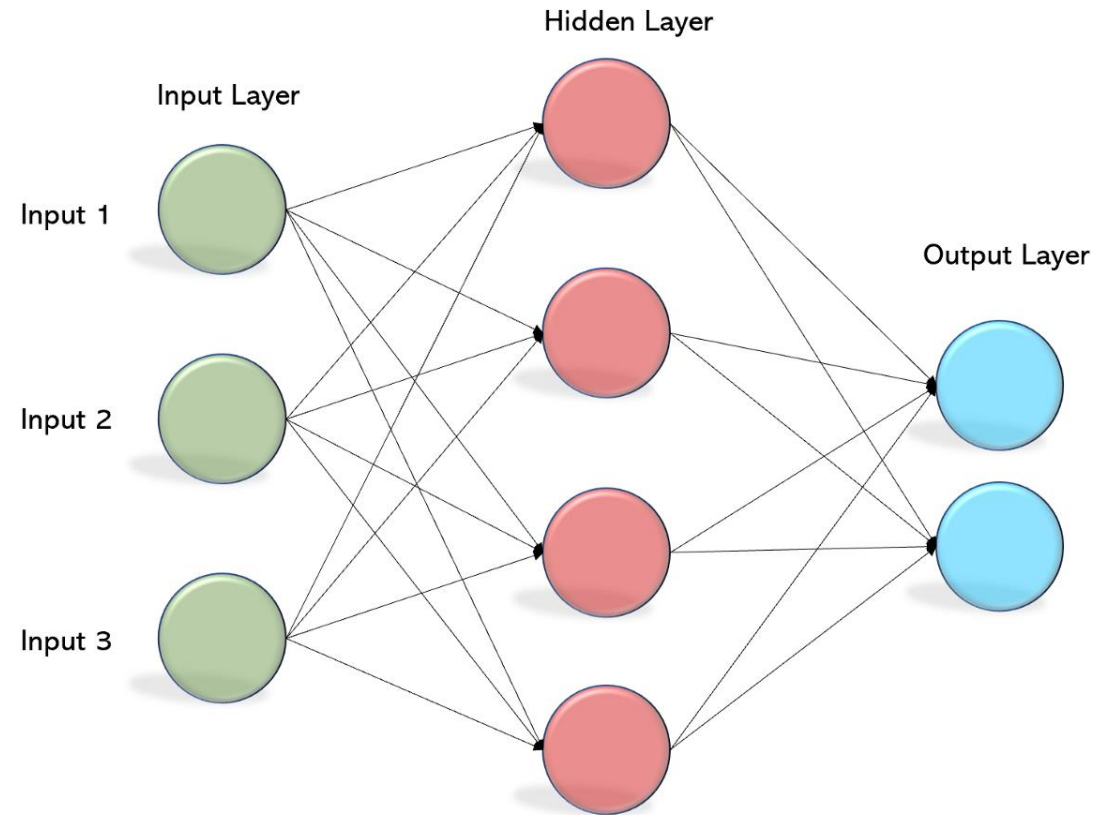
PERCEPTRON

- A perceptron (the blue circle) can be thought of as its own n dimensional linear regression model
- A perceptron takes a weighted sum of its inputs and adds a bias term (exactly like the linear regression model). This allows the perceptron to draw a classification line.
- Formula for perceptron is:
$$(\sum_{i=0}^n x_i w_i) + b$$
- These perceptrons can be stacked in layers to produce a multi-layer perceptron (MLP)



MULTI-LAYER PERCEPTRON

- We know how each of these perceptrons works individually. Thus, we know how the entire network works.
- The key thing to remember is that each of the lines represents a weight.
- The outputs of one layer are multiplied by their weights and then used as inputs to the next layer

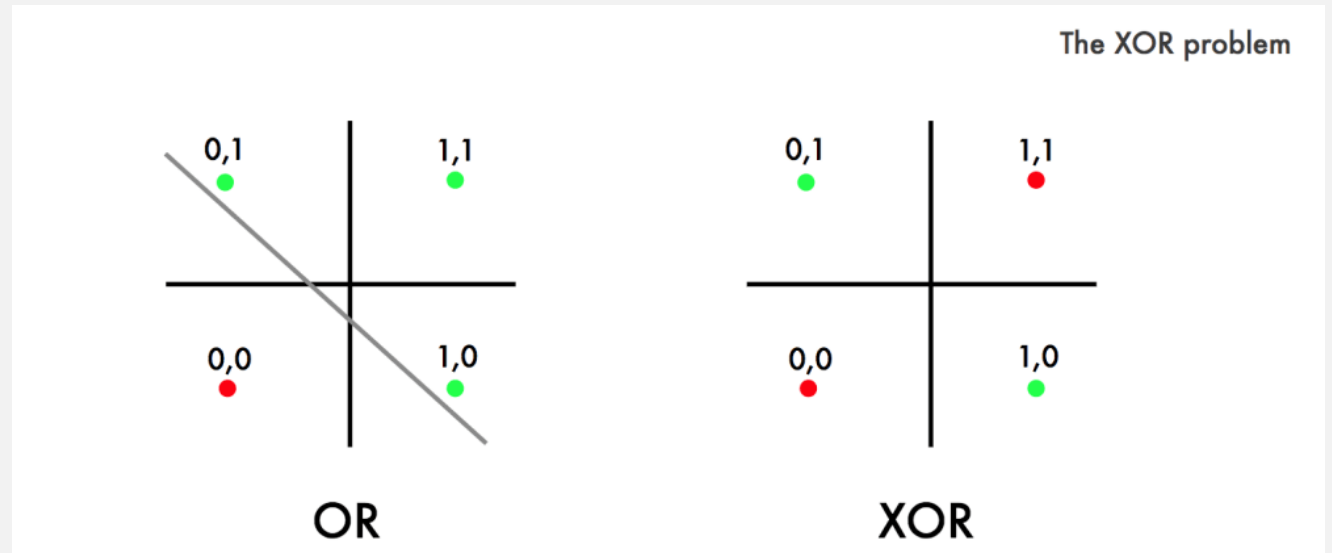


MLP

- Unfortunately, no matter how many layers we use. The MLP will only ever be able to draw a straight line.
- This is a problem as simple problems are unsolvable.
- For example, getting an MLP to classify an XOR gate

MLP

- No matter how you draw the line. You can never draw a single straight line that separates the two classes for the XOR problem.
- This shows that MLP is fundamentally flawed and only able to solve linear problems.

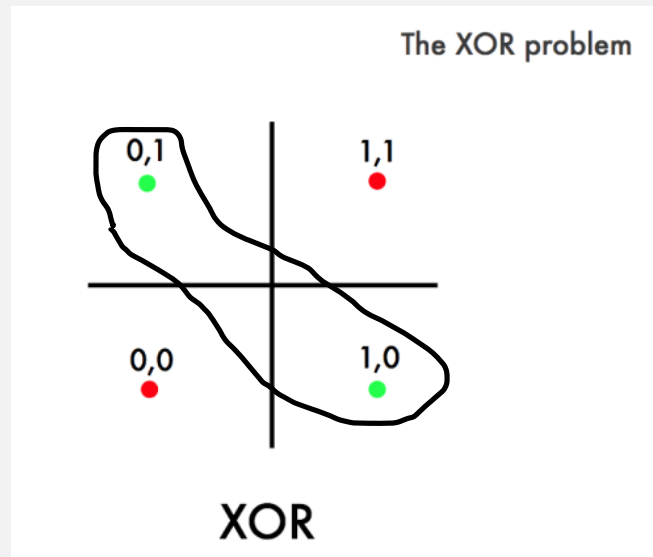


NEURAL NETWORKS

- Thus, we need some way of altering our model of a perceptron to allow to draw lines that are not just linear.
- To do this, we can simply apply an **activation function**. This function must not be linear.
- There are several good choices for activation functions. Let σ denote this function.
- Recall our old formula for the perceptron: $(\sum_{i=0}^n x_i w_i) + b$
- W represents the weights (i.e. the values of the lines) and x represents the inputs from the previous layer. Bias represents the value stored by the neuron.
- We update this formula to the following: $\sigma((\sum_{i=0}^n x_i w_i) + b)$
- This enables our AI to draw lines that are non-linear

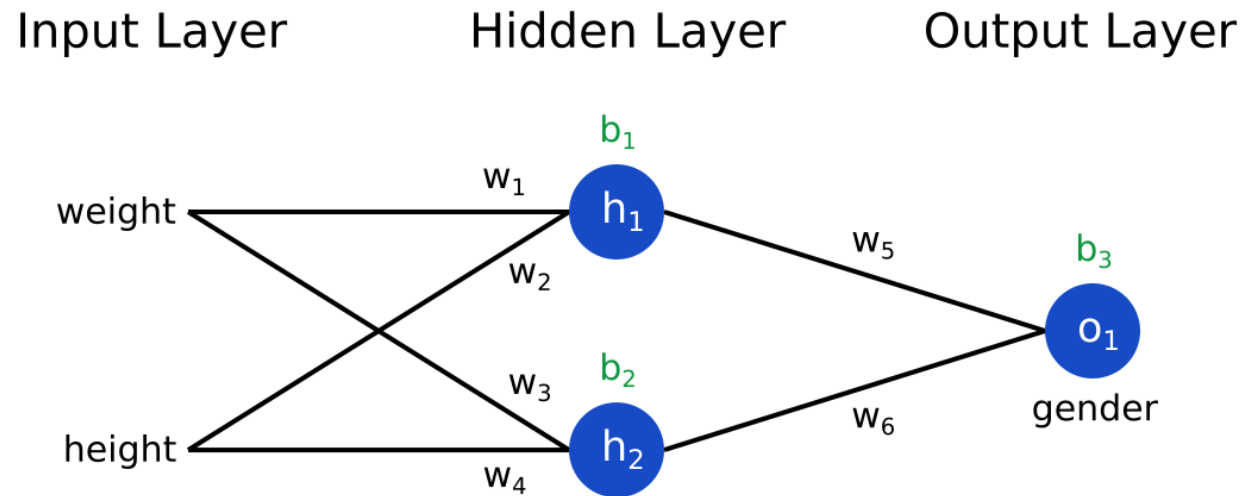
NEURAL NETWORKS

- Using this simple change, our MLP is now said to be a feedforward artificial neural network (ANN)
- This network can draw any line and can solve our XOR problem



NEURAL NETWORKS

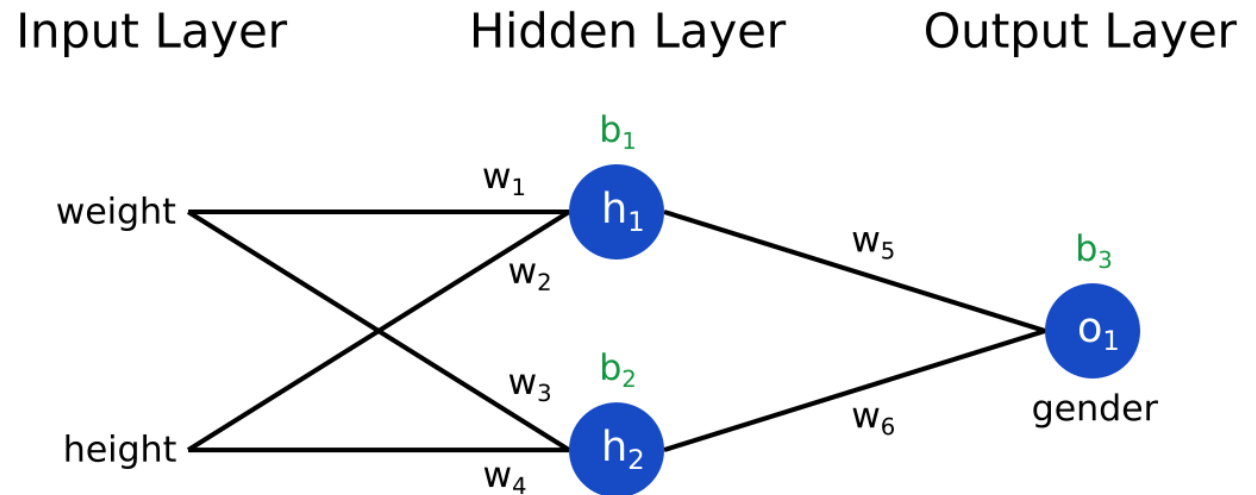
- This is a model that predicts the gender of a person given their height and weight.
- The height and weight values are propagated through the network



NEURAL NETWORKS

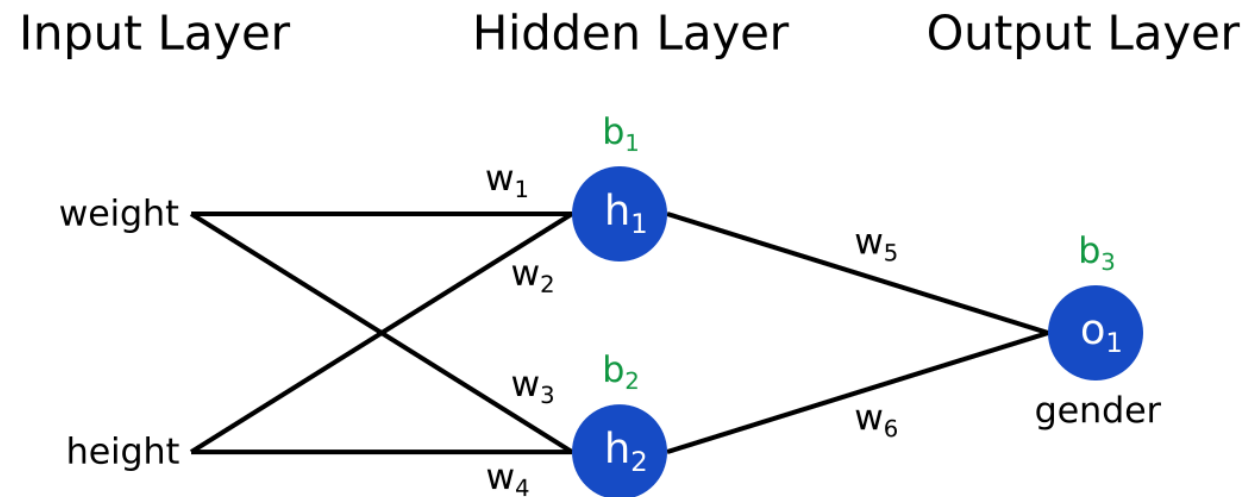
- $inp_1 = weight * w_1 + height * w_2$
- $inp_2 = weight * w_3 + height * w_4$
- $h_1 = \sigma(inp_1 + b_1)$
- $h_2 = \sigma(inp_2 + b_2)$
- $inp_3 = w_5 h_1 + w_6 h_2$
- $o_1 = \sigma(inp_3 + b_3)$
- Thus, we have now propagated our inputs through the network. This is “feed-forward” part of network.
- We might say something like:

$$gender(o_1) = \begin{cases} Girl, & o_1 < 0.5 \\ Boy, & o_1 \geq 0.5 \end{cases}$$



NEURAL NETWORKS

- And remember, we store the values $w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2$ and b_3 .
- These values determine what kind of line the model draws (i.e., how it decides to convert weight and height into gender).



NEURAL NETWORKS

- As neural networks can draw any line, they are able to solve an extremely large range of problems.
- The challenge with neural networks is now finding what combination of weights and biases can be used to draw that line (or sets of lines).
- Unlike linear regression, there is no precise mathematical formula for this problem.
- There are various stochastic techniques used to solve this problem. Most of these attempts involve advanced mathematics and data in order to continuously improve the model
- But it's important to separate the data model from the training method

BACKPROPAGATION

- The challenge with these AI models is choosing the right values for the weights and biases in order to make the most effective neural network possible.
- We will focus on supervised learning techniques for training the model.
- Supervised learning requires a dataset of labelled items. For example, we might have a dataset of images that are labelled as either “cat” or “dog”. We may have a dataset of people’s height, weight and gender.
- Given these datapoints, we can constantly improve the weights and biases of our network to draw a better line

BACKPROPAGATION

- In order to improve our network, we need some mathematical function, l , that describes how close a given network guess is to the correct output.
- Let's use our gender neural network as an example. The output node should be 1 if the height and weight match those of a boy. If it matches a girl it should be 0. Thus, the output can be interpreted as “probability that person is a boy”.
- A common loss function is called mean square error. It takes the magnitude of the error and squares it for all output nodes.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

BACKPROPAGATION

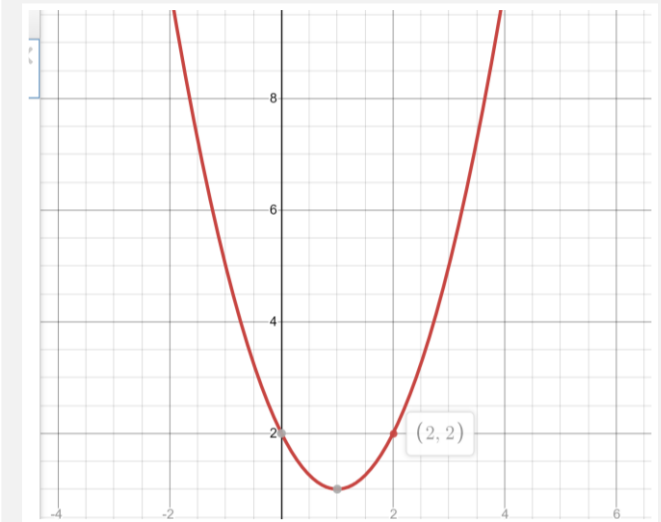
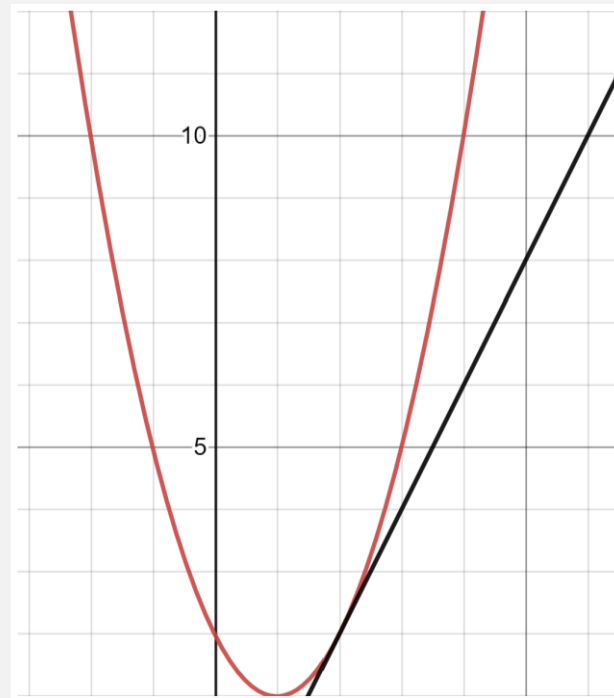
- So, for our specific network, $l = (o_1 - t_1)^2$. Where t_1 is the image label (i.e.,
$$t_1 = \begin{cases} 1, & \text{person is boy} \\ 0, & \text{person is girl} \end{cases}$$
- We know have some function that tells us how wrong / right our network is. We want to minimise this function (get it as close to zero as possible).
- As shown earlier, we know the complete mathematical formula for o_1 , thus we can take the derivate of the function.
- $\frac{dl}{do_1} = 2 (o_1 - t_1)$

BACKPROPAGATION

- As we know all the weights and biases, we can use the chain rule to find $\frac{dl}{dw_i}$ and $\frac{dl}{db_i}$ for all i.
- We won't go too much into the maths (this is just an intro!!!)
- But we calculate the gradient of the loss with respect to each network component and store all these values.

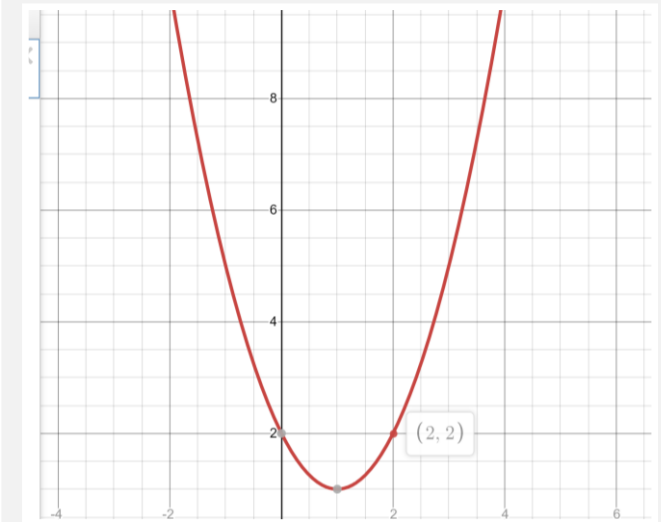
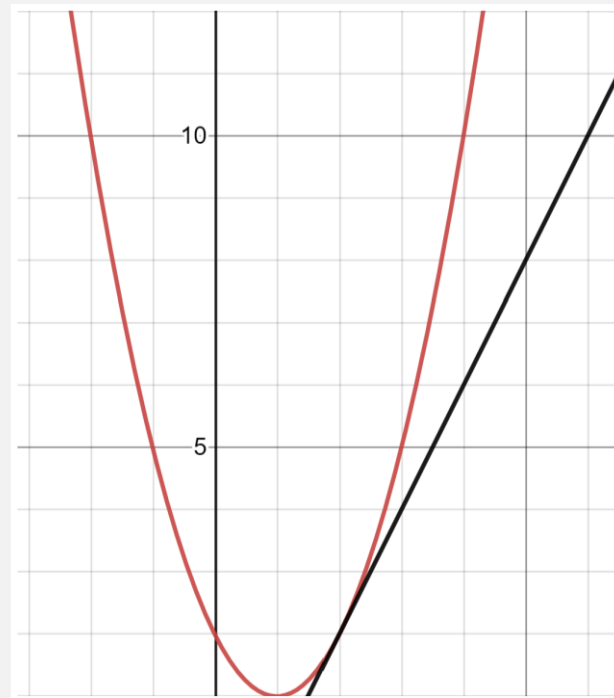
- The gradient represents the direction of steepest ascent at the current point.
- Thus, if we go the other direction, we get the fastest direction of descent.
- The y axis can be thought of as the loss and the x axis can be thought of as the model.
- We want to minimize our loss, so we want to follow the gradient line.

BACKPROPAGATION



- The idea is that we only go a very **small** distance along the gradient line (as the gradient is constantly evolving, the gradient estimate won't be valid for moving large distances).
- The idea is that we repeatedly move at small steps in order to find a **local minimum**. This represents a point where the loss of the model is low.

GRADIENT DESCENT



GRADIENT DESCENT

- Of course, we can't alter o_1 , only the weights and biases. Recall that we calculate $\frac{dl}{dw_i}$ and $\frac{dl}{db_i}$. When these functions have a lower value, it means that they contribute less to the overall error within the network. Therefore, these values should be changed less than those with larger values.
- We update all of our weights and biases using the following formulae
- $w_i = w_i - \frac{dl}{dw_i} * r$
- $b_i = b_i - \frac{dl}{db_i} * r$
- r is known as the **learning rate** and is a small number that decides how large each change to the parameters should be. A normal value for r is 0.001. Larger values lead to faster initial training but make it harder for the network to converge to a minimum

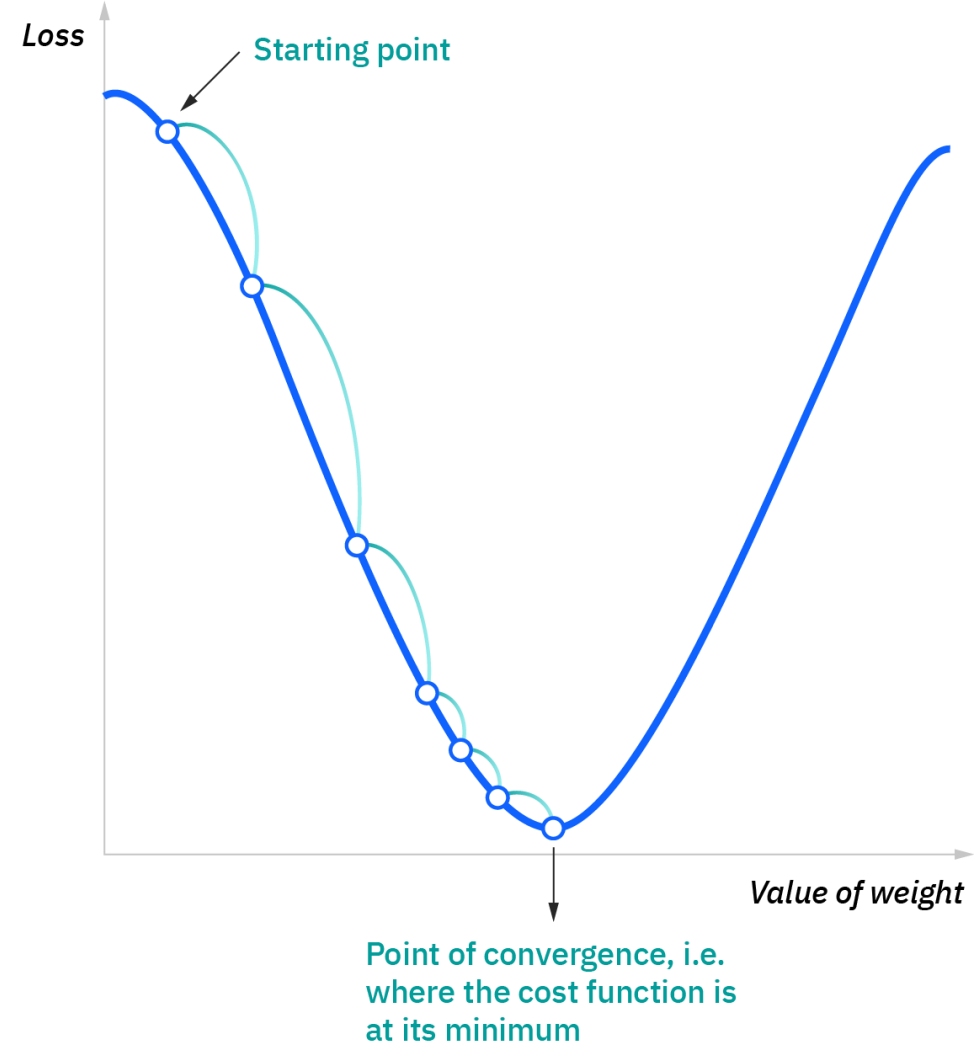
BACKPROPAGATION

- The maths is very complicated, however the overarching idea is that we make small tweaks to each of the parameters to reduce our loss over time
- For each point, we calculate the path that most quickly reduces the loss. We then take a very small step along this path.
- We multiply all weights and biases by the same constant, so weights that are contributing more to the error of the network are changed more than those that are contributing less

GRADIENT DESCENT

- Although, this function looks for local minima, local minima typically turn out to be good indications of global minima. There are advanced techniques to make the system more likely to find the global minimum.
- However, you don't need to understand the maths. All you really need to know is that given a set of labelled data, the neural network can repeatedly apply numerical functions to produce an AI that reduces the loss function (i.e. classifies gender most effectively).

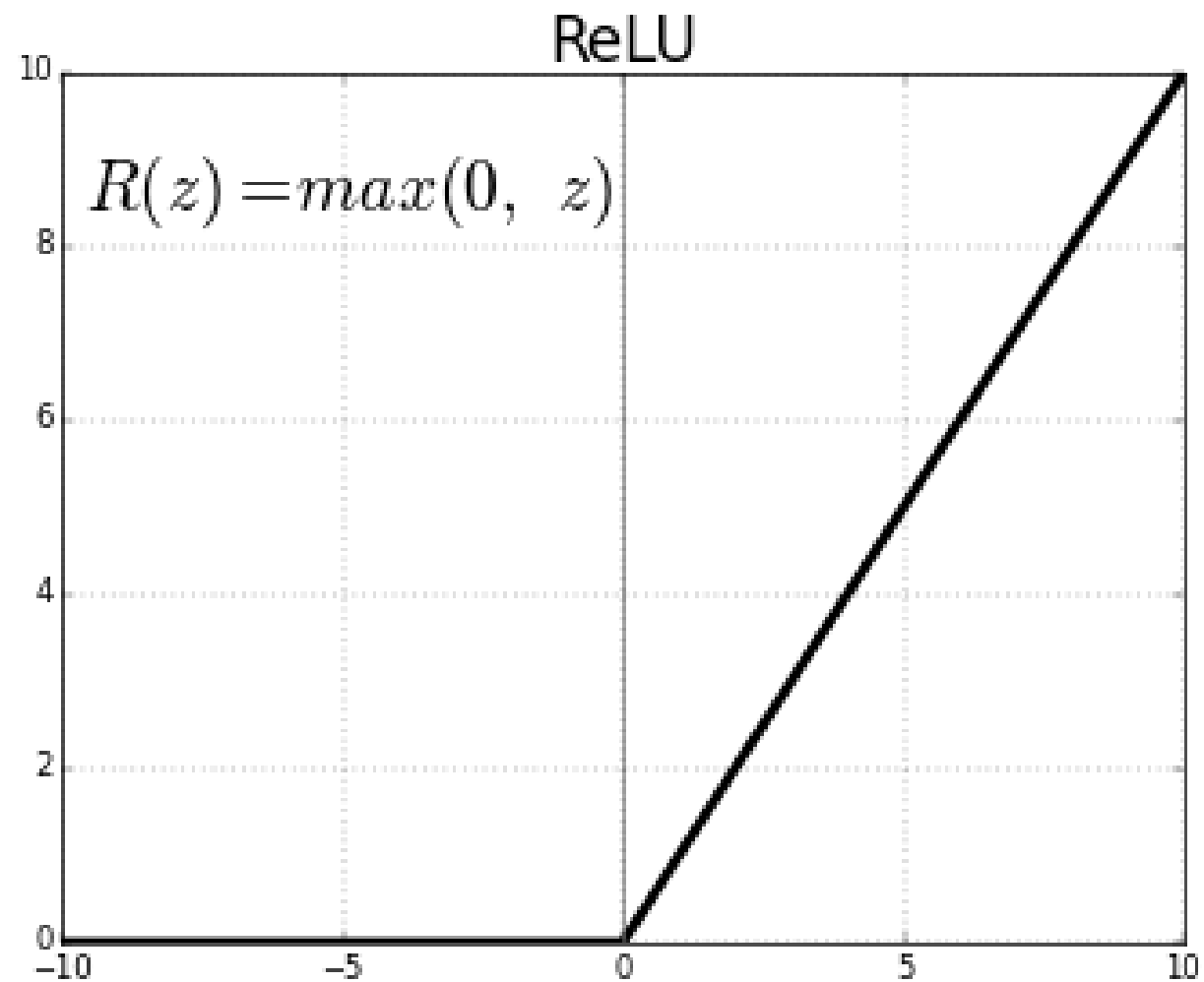
BACKPROPAGATION



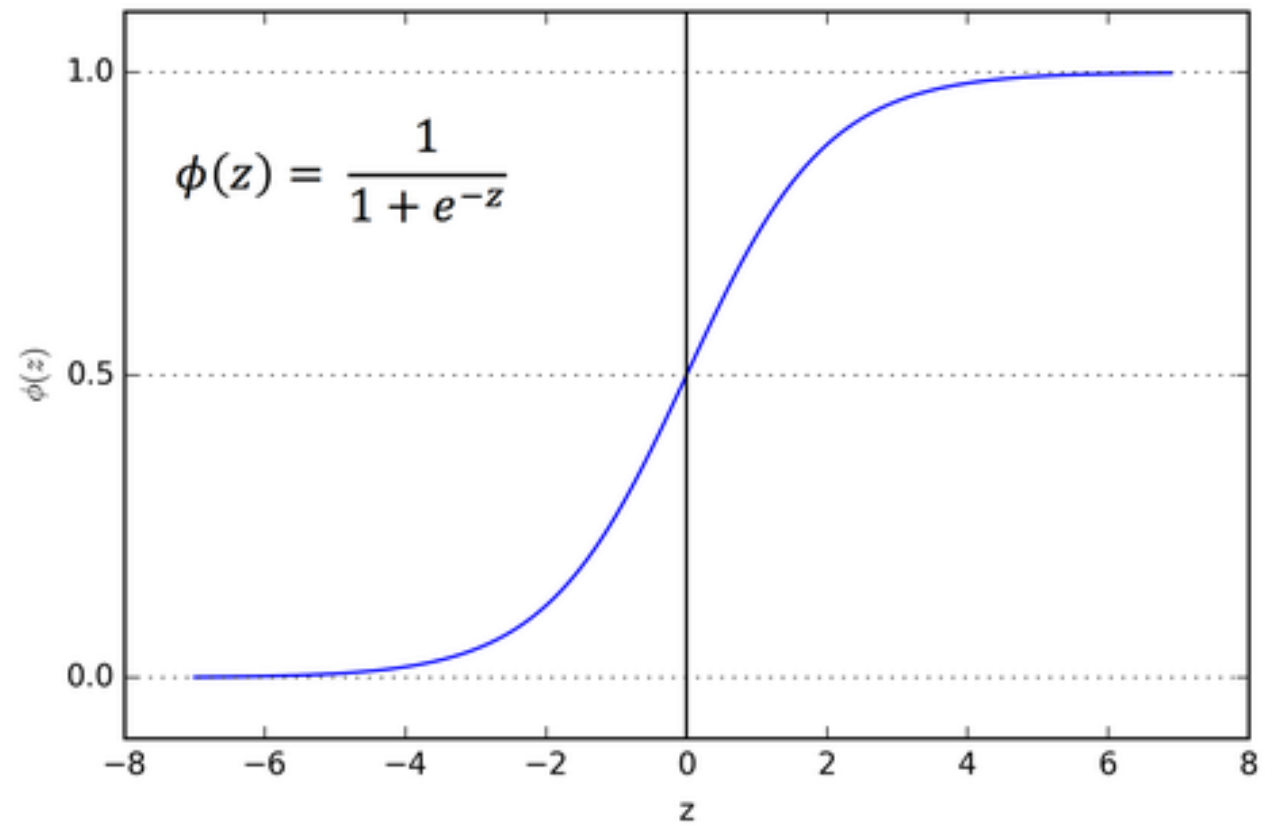
ACTIVATION FUNCTION

- We discussed the need for a non linear function, σ .
- In practice, there are many different functions that are used.
- Arguably the two most common ones are reLu and sigmoid.
- reLu or rectified linear units is the most versatile choice and also cheap to compute.
- Sigmoid also performs very well in classification tasks (along with a function known as softmax)

ReLU



SIGMOID

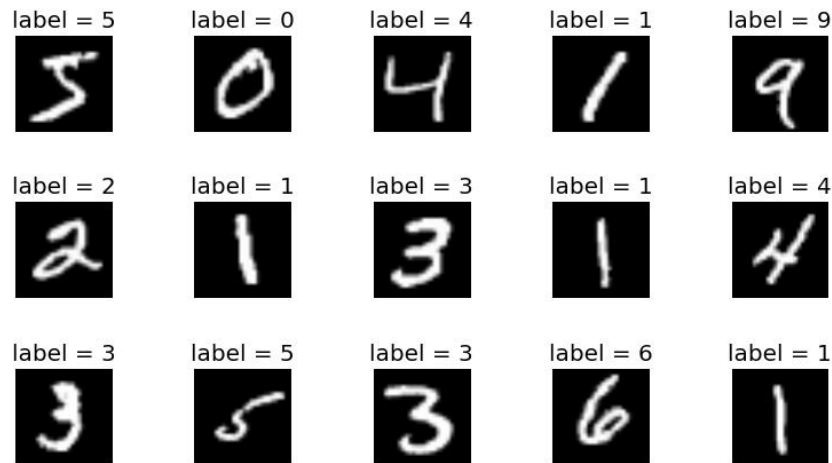


NEURAL NETS

- As already evidenced, there are a large number of programmer defined hyperparameters that differentiate model learning approaches.
- For example:
- The learning rate
- The network architecture (number of layers and number of nodes in each layer)
- The loss function
- The activation function
- There are many more possible hyperparameters for other training schemes
- One of the parameters that we will look at later is the **batch size**

DATA

- Now, we shall go about making our own neural network. This network will look at 28 x 28 pixel images of handwritten digits and try to classify what digit it is.
- To do this using supervised learning, we need a dataset of images. Preferably our dataset will be as large as possible.
- Fortunately, there is a dataset called the MNIST dataset. This dataset consists of 60,000 handwritten digits.



DATA

- In order to train a neural network we typically break our dataset down into three categories.
- Firstly, we will have a training dataset. This is the largest dataset and this is used to train the network. These set of images are used for repeated gradient descent.
- We will have a much smaller dataset called the training set. This set is used to evaluate the performance of the neural network on data it has never seen before.
- We will also have a validation dataset. This serves a very similar purpose to the training set. However, as programmers, we will typically tweak our models after seeing the training score. We only ever use a validation set once – at the very end of training to provide a truly non-biased idea of the AI's performance on data never seen.
- In the case of MNIST we may use the following:
 - 50,000 training images
 - 5,000 testing images
 - 5,000 validation images

DATA

- In order to provide the best estimate for the true loss of the network (and thus the changes that need to be made), we would ideally use all 50,000 images to calculate the gradient every time we update the weights and biases.
- However, in practice this is really computationally expensive. We can make training much quicker by using a much smaller sample each time we update the weights.
- For example, we may only choose to use one image for each weight update.
- This leads to more updates per unit of time and thus the model trains more quickly
- Normally, we like to choose to use multiple images to ensure that the weight updates go in the right direction. If we only use one image, each update will only be focussed on making the network more likely to predict that digit. This may come at the cost of accuracy to other digit predictions

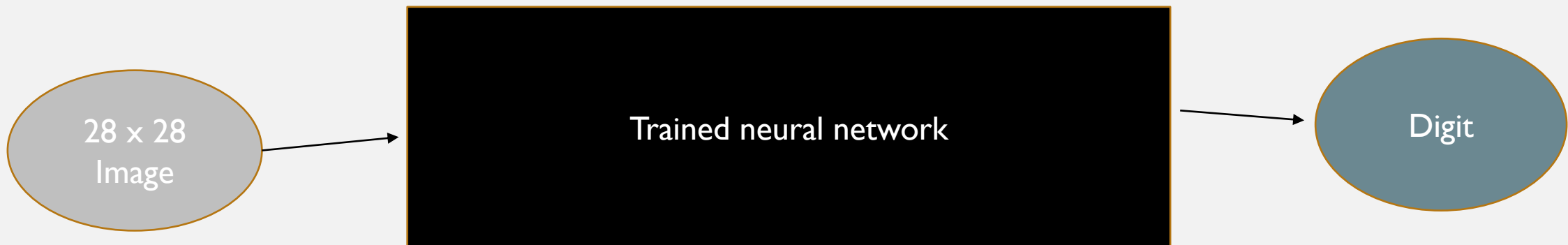
DATA

- This sample size is the **batch size**.
- Normally, we would like to choose a value that strikes a balance between speed and update accuracy.
- For example, we may choose to use 32 – 128 images for the batch size as a fair trade-off.
- Additionally, only using a subset of the training data makes it less likely for the system to get stuck at local minima that are not also global minima.
- The number of **epochs** a program trains for is simply the number of times the AI sees every single picture in the training set.

MNIST AI

- Let's make our own AI that predicts handwritten digits using MNIST!
- This is a task which is basically impossible for a human to ever code, as recognising handwritten digits is too complex to be encoded in a series of instructions.
- Supervised learning is perfectly suited to black box design. Even after the training stage, we still don't understand why the neural net thinks a given image is likely to be a 1. We understand how it has been trained, but not what it does. This makes it a black box

MNIST AI



MNIST AI

- The first thing we need to discuss is the **network architecture**.
- This refers to how many layers our network has and how many nodes should be in each layer.
- Networks with at least two hidden layers, have been shown to be able to emulate any function given enough compute power and size.
- For our network, we need an input layer. This will contain 784 neurons ($28 \times 28 = 784$). Each input node represents the brightness of one pixel in the image. A value of 0.0 means that the pixel is black. 1.0 mean that the pixel is white

MNIST AI

- Next, we need to decide how many **hidden** layers we want to use. For each hidden layer, we need to decide how many nodes we want it to have.
- We want a good network, but we also don't want to make it too big as it will take longer to train and may be too large to be suitable for the dataset size.
- For simplicity, we will use one hidden layer. We will give this layer 128 neurons.
- Finally, we have our output layer. This layer will have 10 neurons. Neuron #0 can be interpreted as “the probability that the image is the digit 0”. Neuron #1 can be interpreted as “the probability that the image is the digit 1”.

MNIST AI

- For our **activation function**, we will use the sigmoid activation for all nodes. This is because sigmoid typically performs well for classification tasks
- As sigmoid produces a value in the range (0.0, 1.0), the interpretation of the output nodes as probabilities is even clearer to see.
- Note how output layer uses one-hot encoding. We want all output nodes except for one to have the value 0. And then we want one to have the value one.
- So for the label one, we would want the following output:
0100000000

MNIST AI

- For our **learning rate** we want a value that leads to good convergence. Thus, we will set it to 0.001. This will lead to fairly small weight and bias updates, making each individual update more accurate. This will make the training time larger but ultimately lead to a stronger model.
- For our batch size, we want to use a large enough volume of images to make each update accurate. However, we also want our model to be trained in a reasonable time so we need to choose reasonable batch size. Let's choose a **batch size** of 128.
- We also need to decide how long to train our model for. We do this by setting the number of **epochs** (the number of times the model sees the entire dataset). In order to make our model training quick, let's choose a value like 5.

MNIST AI

- We have had to specify the following model **hyperparameters**:
- **Learning rate = 0.001**
- **Batch size = 128**
- **Activation function = Sigmoid**
- **Epochs = 5**
- **Network Architecture = 784 – 128 – 10**

MNIST AI

- Clearly, these values are unlikely to be the best possible values. This is one of the tricky things about AI training, it's very hard to actually set these parameters. There's no simple formula that always works.
- Often model training requires lots of trial and error.
- Even with our small model, there's over 100,000 trainable parameters!
- For each batch, we are computing the output for each image. This output is used to calculate the gradient of the loss with respect to each of the 100,000 parameters. This is a lot of computation and takes a lot of time

MNIST AI














- The biggest reason for AI advances seen today is advances in computer hardware. 15 years ago, computers were not powerful enough to perform these expensive computations fast enough to train models.
- Today, hardware is still a big bottleneck. However, a small model such as this one is now relatively cheap to compute.

MNIST AI

- I have developed my own machine learning library entirely from scratch using Java. It implements all of the logic and maths described (plus some more advanced features).
- I have written this library so that it can be applied to the MNIST dataset.
- This can be found at: <https://github.com/philipmortimer/AI-Course/tree/main/Programs/Part%204/Artificial-Neural-Network-MNIST>
- Please feel free to download it and use my library to make your own model

MNIST AI

- Inside the folder, select the “.jar” file to see a pretrained neural network in action. This one uses a large architecture and has been trained for a long time

	.git	08/04/2022 20:29	File folder	
	nbproject	08/04/2022 20:27	File folder	
	src	08/04/2022 20:27	File folder	
	bestNetstruct.txt	08/04/2022 20:27	Text Document	33,759 KB
	build.xml	08/04/2022 20:27	XML Document	4 KB
	dataProcessedForGitHubSizeLimit.txt	08/04/2022 20:27	Text Document	1 KB
	Handwritten_digit_recog_MNSIT.jar	08/04/2022 20:27	Executable Jar File	16,356 KB
	manifest.mf	08/04/2022 20:27	MF File	1 KB
	mnist_test.csv	08/04/2022 20:27	Microsoft Excel Co...	17,875 KB
	networkInfo.txt	08/04/2022 20:27	Text Document	0 KB
	one.txt	08/04/2022 20:27	Text Document	53,542 KB
	README.md	08/04/2022 20:27	MD File	5 KB
	two.txt	08/04/2022 20:27	Text Document	53,529 KB

MNIST AI

- Open the project in either IntelliJ or NetBeans (recommended to use the NetBeans IDE).
- The only bit of code we need to worry about is “HandwrittenDigitRecogMNSIT.java”. This contains the code snippet that interacts with the neural network library to train the model.

```

public class HandwrittenDigitRecogMNSIT {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        DataProcessForGitHub.processDataForGitHubFileSizeLimit();
        NeuralNetwork ne=new NeuralNetwork(new int[]{784,128,10});
        double learningRate=0.001;int noOfEpochs=5;int batchSize=128;
        int totalSizeOfTrainingData=60000;int printAfterNoIterations = 468/2;boolean loadTraingingDataInMemory=true;
        ne.trainNew(learningRate, noOfEpochs, batchSize, totalSizeOfTrainingData,printAfterNoIterations,loadTraingingDataInMemory);
        MenuUI me=new MenuUI();
        me.init(ne.copyNetwork());
        me.setVisible(true);
    }
}

```

- “learningRate” sets the learning rate. This is currently set to 0.001
- “noOfEpochs” dictates how many epochs the network trains for. This is the number of times each image is seen by the network. This is roughly analogous to the time spent training. I’ve set it to 5, to ensure a short training time
- “batchSize” alters the number of images used in each weight update calculation. It’s currently 128
- The line “NeuralNetwork ne = new NeuralNetwork(new int[]{784, 128, 10});” initialises a neural network.
- To change the architecture, simply change the array passed as an argument. In this case, the array means we have 784 input neurons, one hidden layer containing 128 neurons and then 10 neurons in the output layer.
- If we wanted to change the architecture we could do something like: “NeuralNetwork ne = new NeuralNetwork(new int[]{784, 200, 100, 10});”
- You don’t need to worry about **any** of the other pieces of code

TRAIN + EVAL

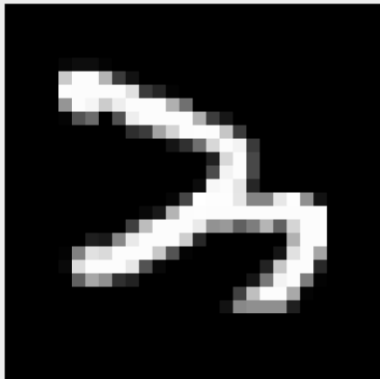
- Let's train the model!
- We see that our model achieves an accuracy of ~97%
- The pretrained model achieves an accuracy of 98.99%.
- This is extremely impressive and shows just how effective supervised learning is.

THE WRONG ONES

- Our neural network classifies the vast majority of digits correctly, far better than any hard-coded program could ever achieve.
- However, even when it guesses the digits wrong, it does so on extremely badly written digits. And it also gives less confidence to these incorrect guesses

THE WRONG ONES

- The Computer guesses that this is a two.
- The correct answer is a three
- However, this is a badly written digit and could actually plausibly be a quickly written two in my opinion



Overall Accuracy Of Network (tested across whole data set) (%)

98.99

Total Images Viewed: 19

Total computer got correct: 18

Correct so far (%): 94.73

Confidence (%) that image is number

0	1	2	3	4	5	6	7	8	9
1.152	2.105	99.99	4.804	4.948	2.583	1.994	1.418	4.869	2.617

Menu

Confidence in answer(%) - higher percent = more confident

99.99951937113694

Computer Guess: 2.0

Correct Answer: 3.0

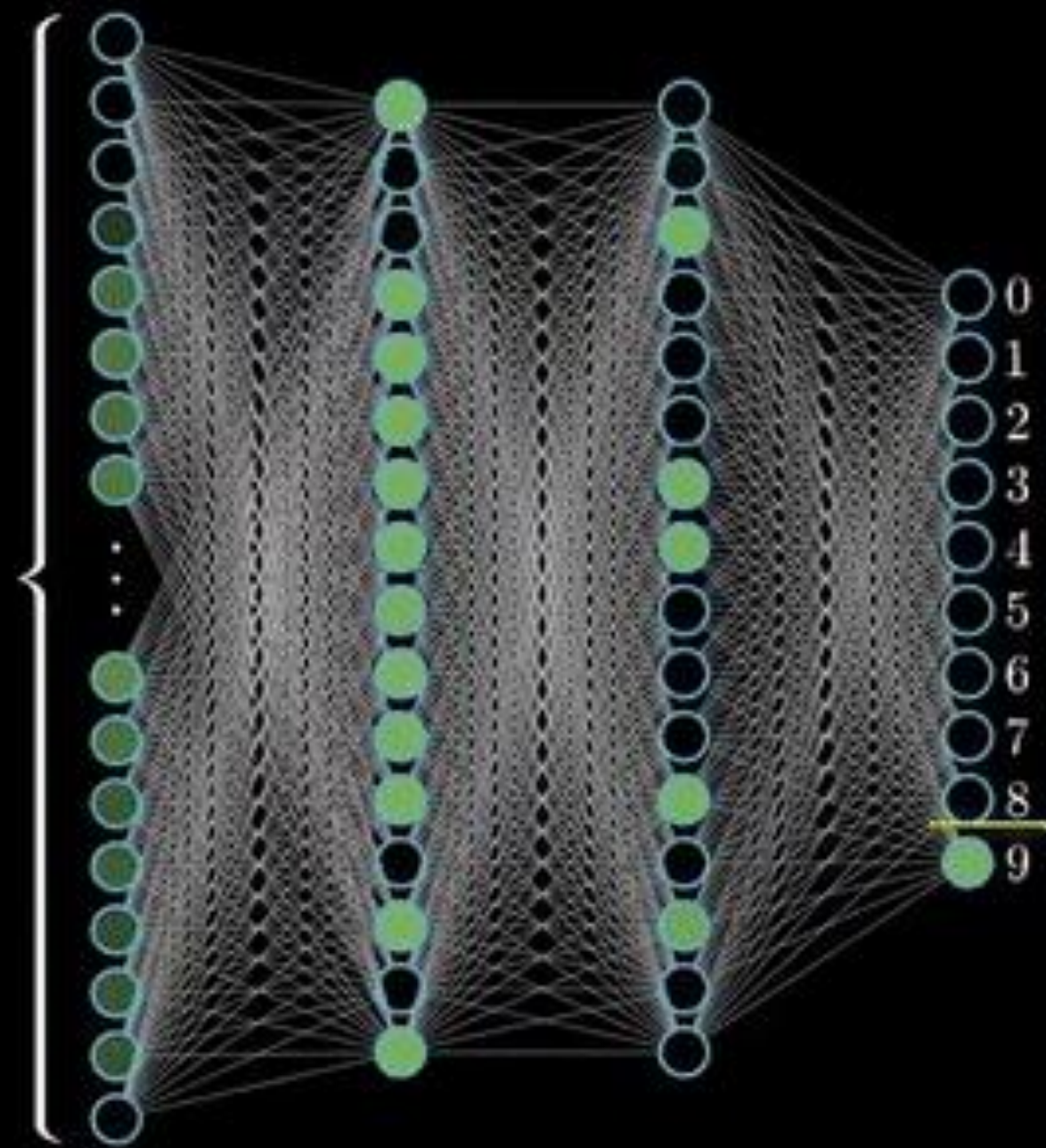
Next Image

AI

- This shows us that machines are extremely intelligent and can learn very effectively
- Supervised learning techniques are often limited by human error when it comes to labelling
- Note, that training a quick network still took a fair bit of time. This shows that the calculations performed in ML process are extremely expensive and require strong hardware.



784



SUMMARY

- We discussed neural networks
- Neural networks are stacked layers of perceptrons that use non-linear functions
- Neural networks are trained to draw a number of lines through a dataset in order to categorise items
- We looked at how an AI can be trained to recognise handwritten digits, something that is impossible for hard-coded systems
- We trained our own AI that does exactly that
- It's important to remember that this is a very difficult subject and that you are unlikely to understand a lot of things talked about today
- However, you now should have a intuition that neural networks are black box functions that, given a labelled dataset and lots of compute time, can solve incredibly complex problems with a human like level of intelligence.

QUESTIONS

ANY
QUESTIONS?



FURTHER READING

- There are many excellent resources about Neural Networks and machine learning to be found on the web.
- However, I would strongly recommend this video series:
https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6RI_67000Dx_ZCJB-3pi