Intro to Deep Learning Neural Networks

What is Deep Learning?

(don't worry, it's not that deep)

Deep Learning

It is a type of machine learning where multiple layers of processing are used to extract progressively higher level features from data

Content

- Why Deep Learning is better than conventional ML models
- The fundamentals of a Neural Network
- Implementation of a Neural Network

Conventional problems

The basis of all data science is the underlying trust that even seemingly random occurences in the real world have a (complex) pattern inside them. The purpose of all that we are learning here is to learn to recognise those patterns, exploit them and make useful predictions about the future.

Early solutions involved extracting features out of data to use a Regression model (what does that mean?)

A major issue is that it is quite hard to find meaningful features in an image (the features should remain the same when irrelevant changes are made to the image and should change when relevant changes are made)

CONVENTIONAL PROBLEMS

Consider the images below and let us take a moment to appreciate how amazing it is that our brain can classify the first three images as motorcycles whereas the last one as a house. What's actually common between all the three motorcycle pictures?

Conventional Problems

Scalability:

A reasonable assumption is that if we have more data, our prediction will be equally better. Studies have shown that that is not true: after a large amount of data "non-deep" models begin to saturate and do not improve their perfomance by a lot.

One Learning Algorithm Hypothesis

Hence the majority of the work done was only feature extraction (how can we represent this image in a unique "feature vector" such that only similar images give a positive test and all others give a negative test?)

Tens of thousands of researchers would extract features from a data, and this had to be done for all types of classification (because every case is starkly unique) - very time consuming

$\longrightarrow$

Representing the image's essential qualities as a feature vector

$\longrightarrow$

Insert into standard Regression model

slido

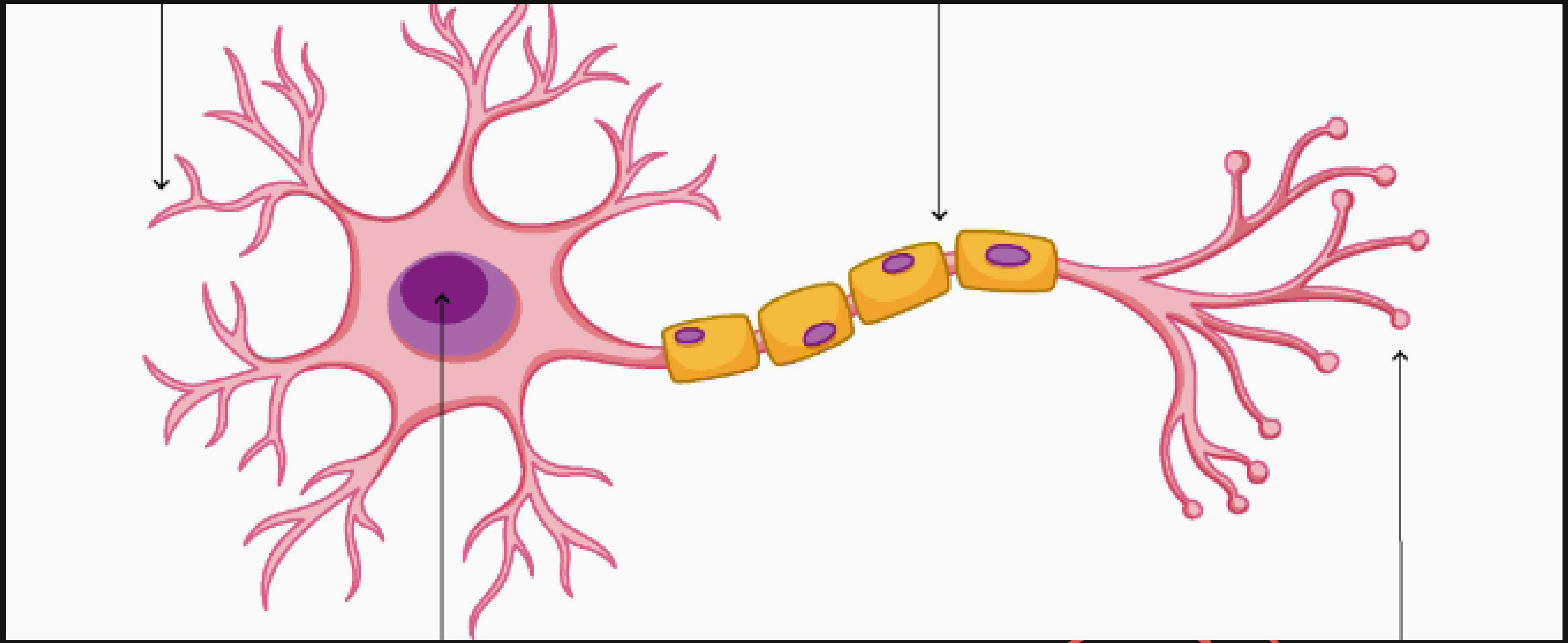
Join at

slido.com

#6047 438

Building a NN: Neuron

Let's turn our attention to what a Neural Network is: the basic structure of any deep learning algorithm. First: what is a neuron?



You can think of a neuron in two ways:

1. something that takes in a bunch of inputs and sends out an output (for those familiar with calculus this is basically what a function does)
2. an empty box which can contain a number - called its activation

What kind of a Network

So these "neurons" must somehow work together to mimic human discrimination. We want to connect these neurons so that they can transmit information between one another.

Furthermore we want to assign to each connection a "weight" so that the network can distinguish "important" transmission of information and "useless" transmission of information.

WEIGHTS AND BIASES

That's exactly what we do! We're almost halfway done...

To be more concrete, a neuron has multiple inputs coming in, each associated with a given weightage, called a "weight". This neuron's activation is simply the weighted sum of all these inputs! Oh, but wait - sometimes different parameters may have different degrees of importance. We want to correct for this by setting a bias.

$$z = f(b + x \cdot w) = f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

Activation Functions

In addition, we often want to keep these calculations we are doing from exploding - we want to contain them. Another reason is that we want to introduce a degree of nonlinearity to include the complex relations between data. So we apply a function on this sum - called an Activation Function.

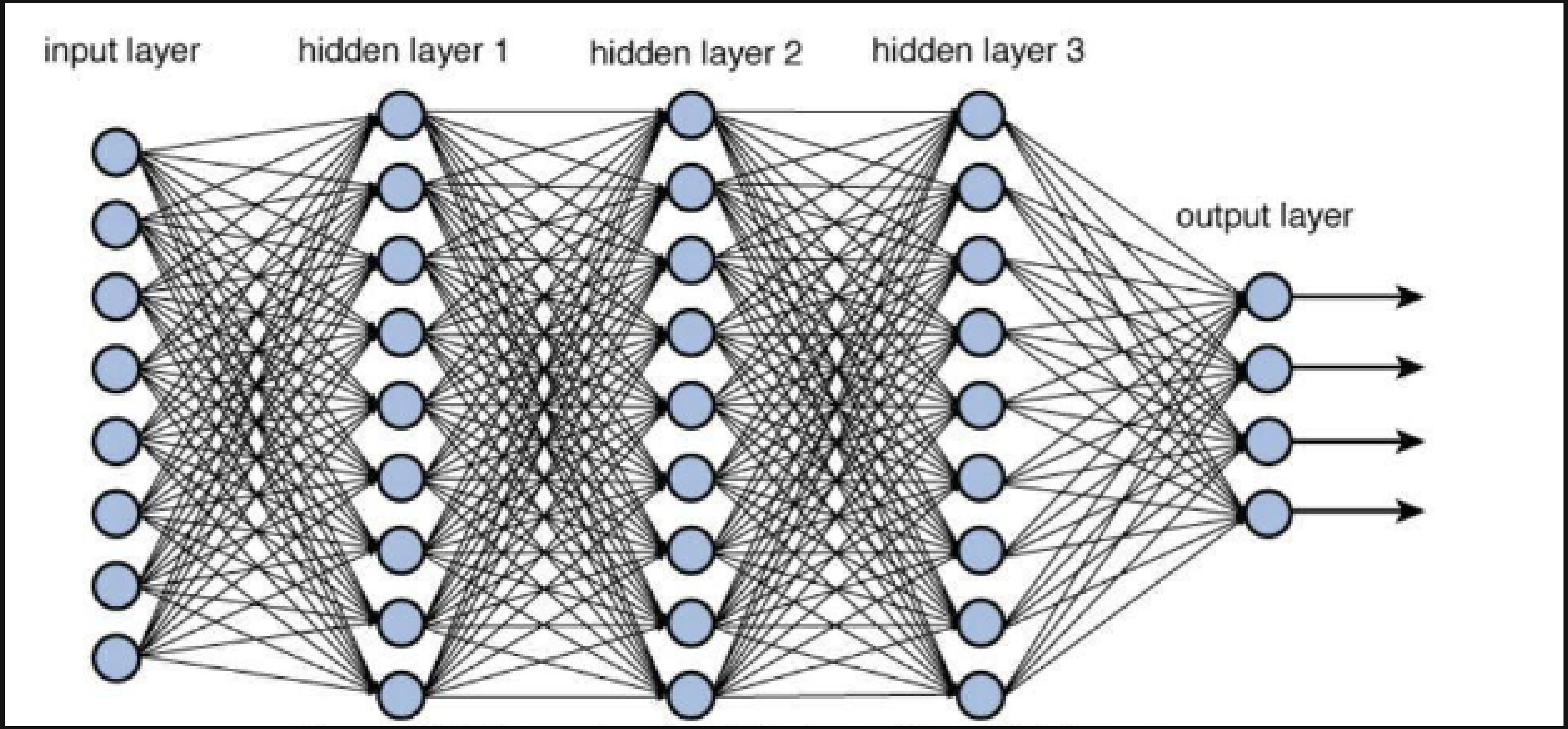
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Different types of activation functions used are

- Sigmoidal activation
- Hyperbolic Tangent activation
- Rectified Linear Unit activation (ReLU)

Out of these ReLU is used the most extensively, gives the fastest results (no computing complicated derivatives) and also solves the vanishing gradient problem.

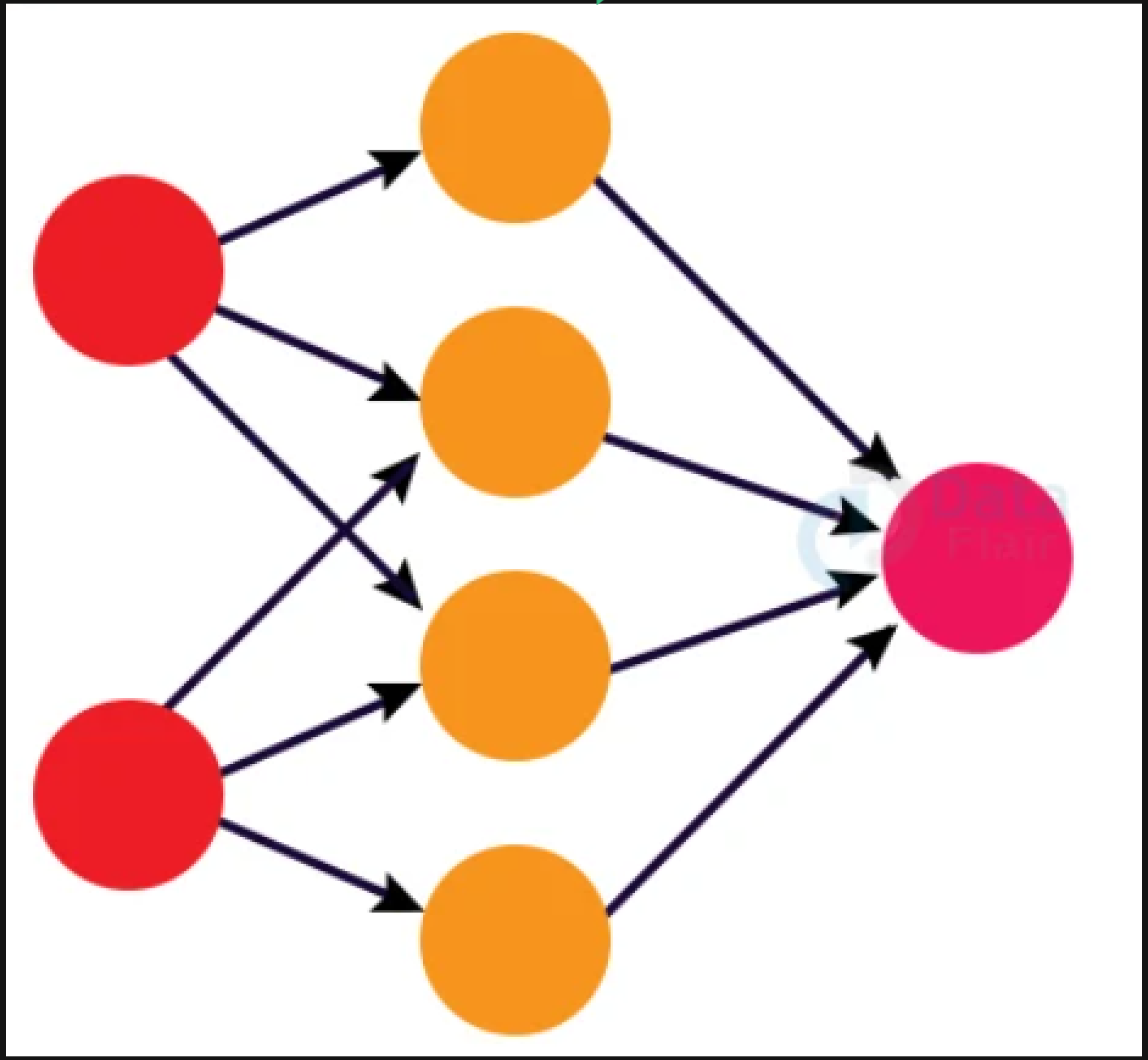
Neural Networks: An Overview



Neural networks are organised into layers as shown. Each circle represents a neuron, each line represents a connection with a weight and the biases and activation functions are not shown. The layers in the middle process more abstract features of the data.

Working of a Neural Network

Okay then. How do we design this to actually do what we want it to do?



We want information to dissolve into its most basic components - it is these "features" which truly distinguish something from others. So these weights dictate what neurons are activated more depending upon the input. This is called a forward pass.

Forward Pass

Say we want to recognise an image. Each image has pixels which contain values that dictate how bright it is (say greyscale for simplicity). Send these as inputs and transfer the input through the network till it produces an output.

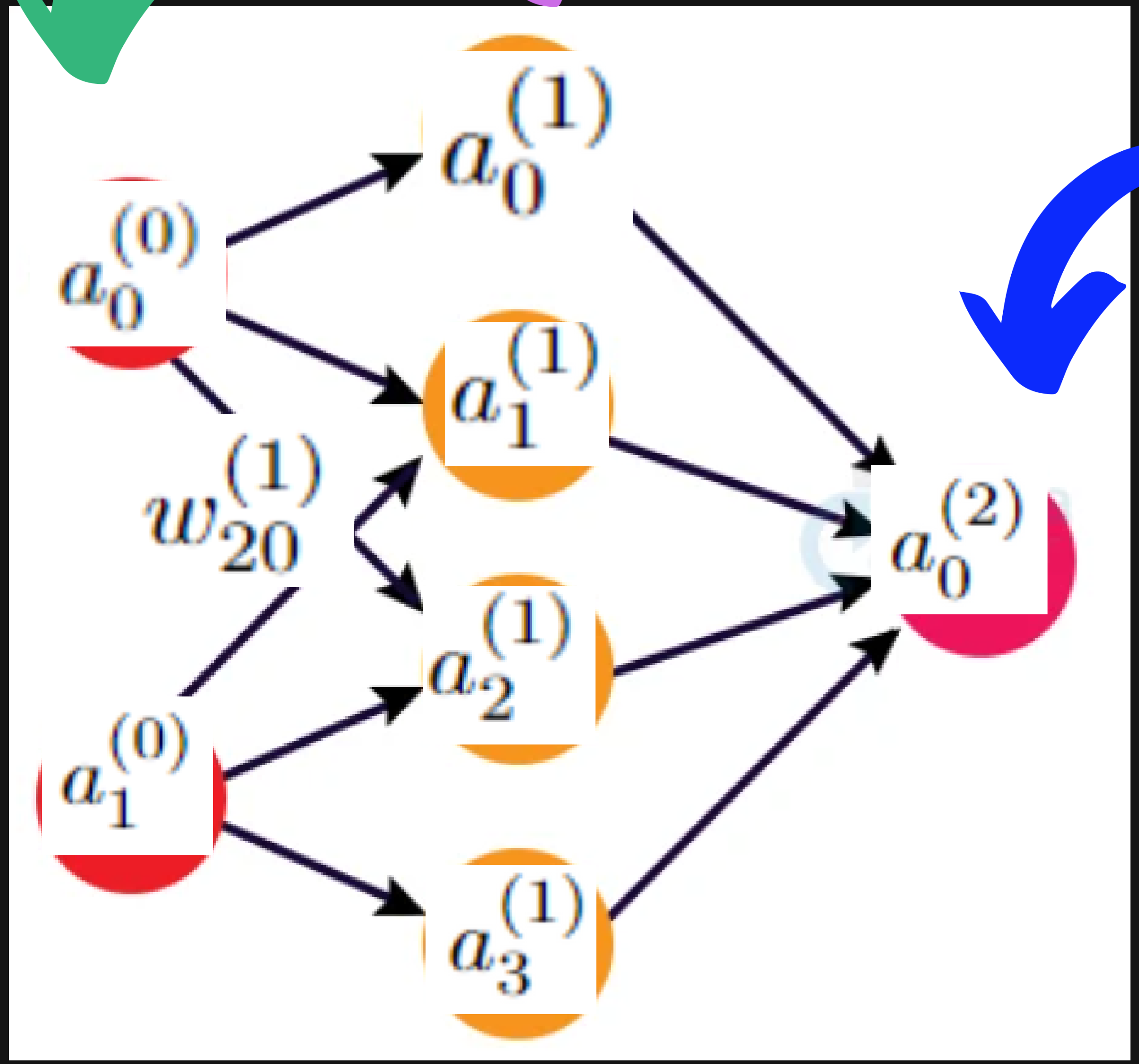
The output we get will be in the form of each of the output neurons having a value each, and it is quite intuitive to say that the neuron with the greatest value is the network's opinion of what class the image belongs to.

Note: at this stage since all the parameters of the network are random we expect a random outcome which is of no use to anyone. Aw! How do we improve?

Layer 0: Input layer

Layer 1: Hidden layer

Layer 2: Output layer



Notation in a Neural Network

$$a_0^{(1)} = a_0^{(0)} w_{00}^{(1)} + a_1^{(0)} w_{01}^{(1)} + b_0^{(1)}$$

$$a_1^{(1)} = a_0^{(0)} w_{10}^{(1)} + a_1^{(0)} w_{11}^{(1)} + b_1^{(1)}$$

$$a_2^{(1)} = a_0^{(0)} w_{20}^{(1)} + a_1^{(0)} w_{21}^{(1)} + b_2^{(1)}$$

$$a_3^{(1)} = a_0^{(0)} w_{30}^{(1)} + a_1^{(0)} w_{31}^{(1)} + b_3^{(1)}$$

$$a_0^{(2)} = a_0^{(1)} w_{00}^{(2)} + a_1^{(1)} w_{01}^{(2)} + a_2^{(1)} w_{02}^{(2)} + a_3^{(1)} w_{03}^{(2)} + b_0^{(2)}$$

....and of course an activation function applied to each of these neurons

Where the "Learning" comes in

The forward pass that we have seen until now is basically what a baby does: random actions which seem childish - I suppose that is where the phrase comes from. We also know that over time we "learn". What's really happening in our brains? Arguably the most important part of a neural network....

Scan this QR Code to mark your attendance for this session

The Cost Function

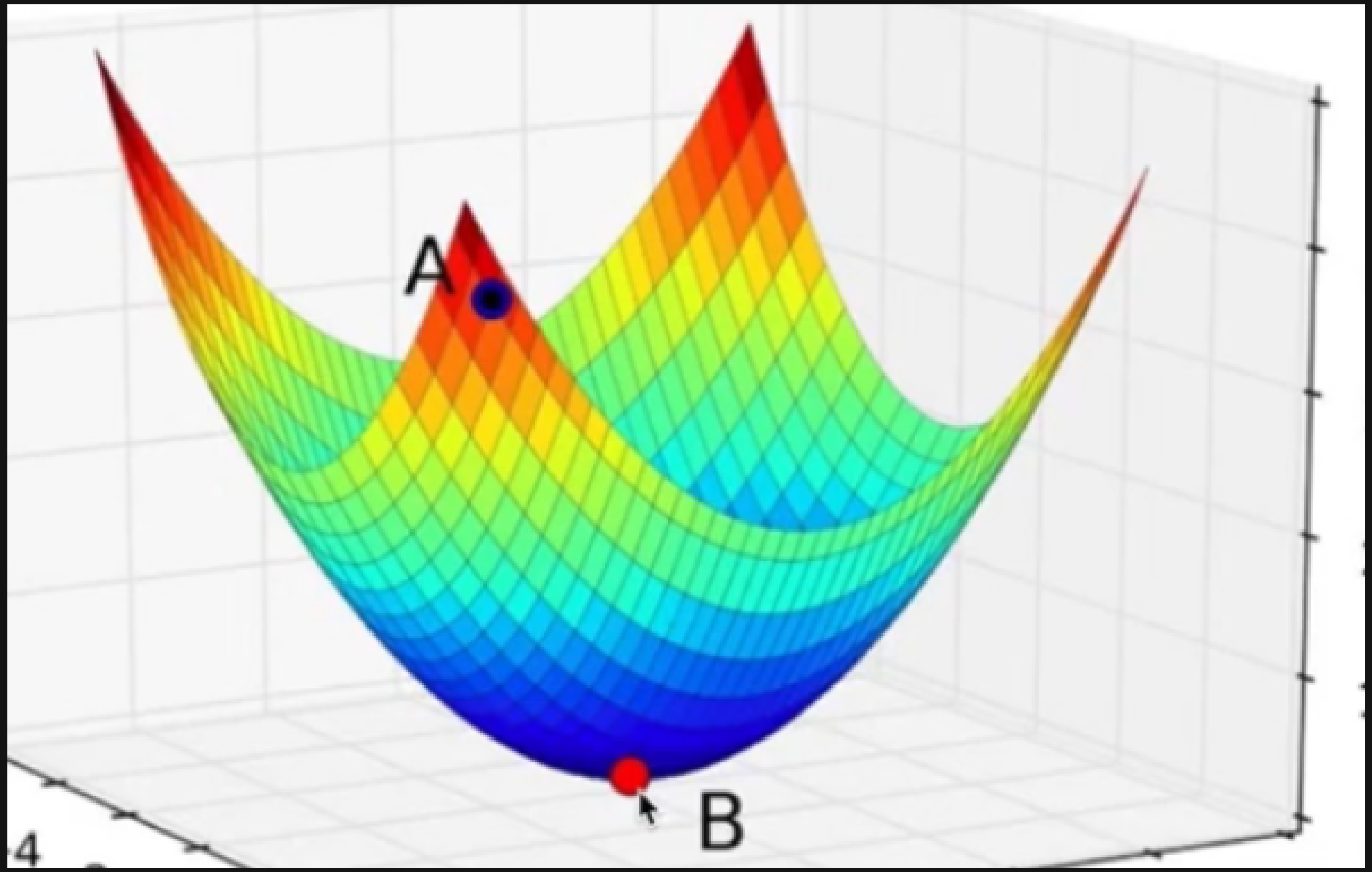
We need to first have a measure for how terrible our network is doing: a sense of how far we are from accurate prediction. We call this the cost function. There are many ways to represent the cost:

1. Mean Squared Error (MSE)
2. Cross- Entropy Loss
3. Mean Absolute Error
4. Hinge Loss

$$C = \frac{1}{N}\sum_{0}^{N-1}(a_i^{(2)} - \hat{y}_i)^2$$

So what is this a function of? It depends only on the weights and biases: the parameters of the network and its output is one real number

The Backward Pass

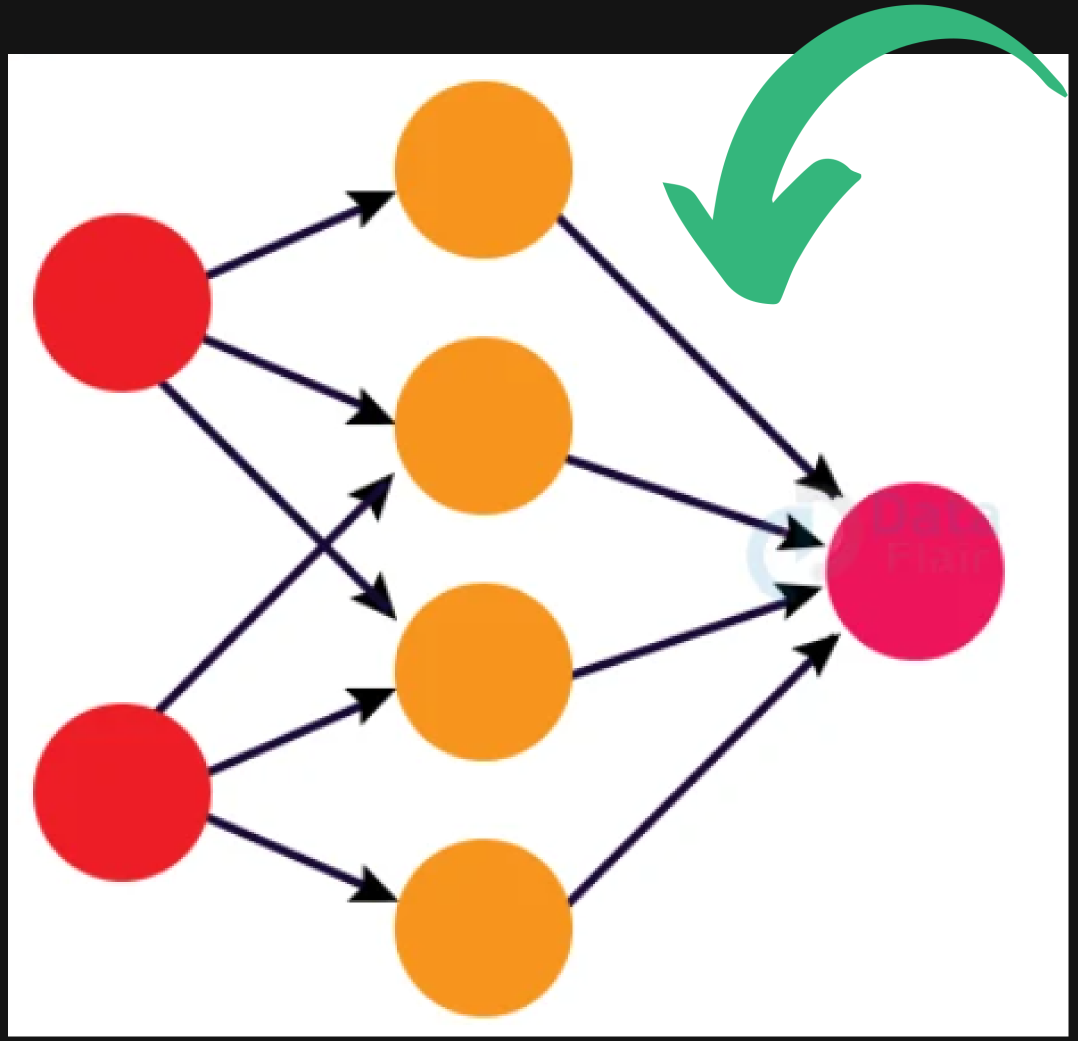


What's our goal here? Clearly the higher the cost function is, the worse our prediction is. This means that we want to minimise the cost function.

Since the cost function is a function of the weights and biases this is equivalent to asking: how can I change the weights and biases such that my cost function decreases?

Backward Pass - The Intuition

Consider the weight shown in the neural network. How does its value affect the cost function C? Play around with it! Increase it a little! Is your cost function decreasing? Maybe it doesn't.



Oops - the above experiment is telling you that you've done the wrong change to that weight to decrease the cost function - which is what you want, after all. So you decrease the weight this time! Voila, it turns out that that is what you need to do. That's all backpropogation is.

Backward Pass - The Mathematics

Now that we've got the intuition behind us, let's get down to the detail - after all as it is said - the beauty lies in the detail.

$$\frac{\partial C}{\partial w_{jk}^{(L-1)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C}{\partial a_j^{(L)}}$$

$$\frac{\partial C}{\partial a_k^{(L-1)}} = \sum_{j=1}^{N_L} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C}{\partial a_j^{(L)}}$$

A good way to find out how much the cost changes as a consequence of the change in a weight is by the partial derivative. In a deep network the weights connecting the previous layers affect the cost to, albeit indirectly. Calculus gives us a nice way to measure the effect of indirect changes: the chain rule.

Using the Chain Rule

$$C = \frac{1}{N}\sum_{0}^{N-1}(a_i^{(2)} - \hat{y}_i)^2$$

$$C = \frac{1}{N}\sum_{i=0}^{N-1}(a_0^{(1)}w_{i0}^{(2)} + a_1^{(1)}w_{i1}^{(2)} + a_2^{(1)}w_{i2}^{(2)} + a_3^{(1)}w_{i3}^{(2)} + b_0^{(2)} - \hat{y}_i)^2$$

$$\frac{\partial C}{\partial w_{00}^{(2)}} = \frac{\partial C}{\partial a_0^{(2)}}\frac{\partial a_0^{(2)}}{\partial w_{00}^{(2)}} = \frac{2}{N}(a_0^{(1)}w_{00}^{(2)} + a_1^{(1)}w_{01}^{(2)} + a_2^{(1)}w_{02}^{(2)} + a_3^{(1)}w_{03}^{(2)} + b_0^{(2)} - \hat{y}_0) \times a_0^{(1)}$$

$$\frac{\partial C}{\partial w_{00}^{(1)}} = \frac{\partial C}{\partial a_0^{(2)}}\frac{\partial a_0^{(2)}}{\partial a_0^{(1)}}\frac{\partial a_0^{(1)}}{\partial w_{00}^{(1)}} = \frac{2}{N}(a_0^{(1)}w_{00}^{(2)} + a_1^{(1)}w_{01}^{(2)} + a_2^{(1)}w_{02}^{(2)} + a_3^{(1)}w_{03}^{(2)} + b_0^{(2)} - \hat{y}_0) \times w_{00}^{(2)} \times a_0^{(0)}$$

$$\frac{\partial C}{\partial b_0^{(1)}} = \frac{\partial C}{\partial a_0^{(1)}}\frac{\partial a_0^{(1)}}{\partial b_0^{(1)}}$$

Gradient Descent

From the previous slide we can calculate the changes in the cost function with respect to the changes in the weights and the biases of the network. It makes sense to adjust the weights that way such that the cost decreases.

The direction of the steepest ascent of a function is given by the gradient of the function. Since we want to descend in the steepest possible way (minimise the cost function the fastest?) we proceed along the direction of the negative gradient.

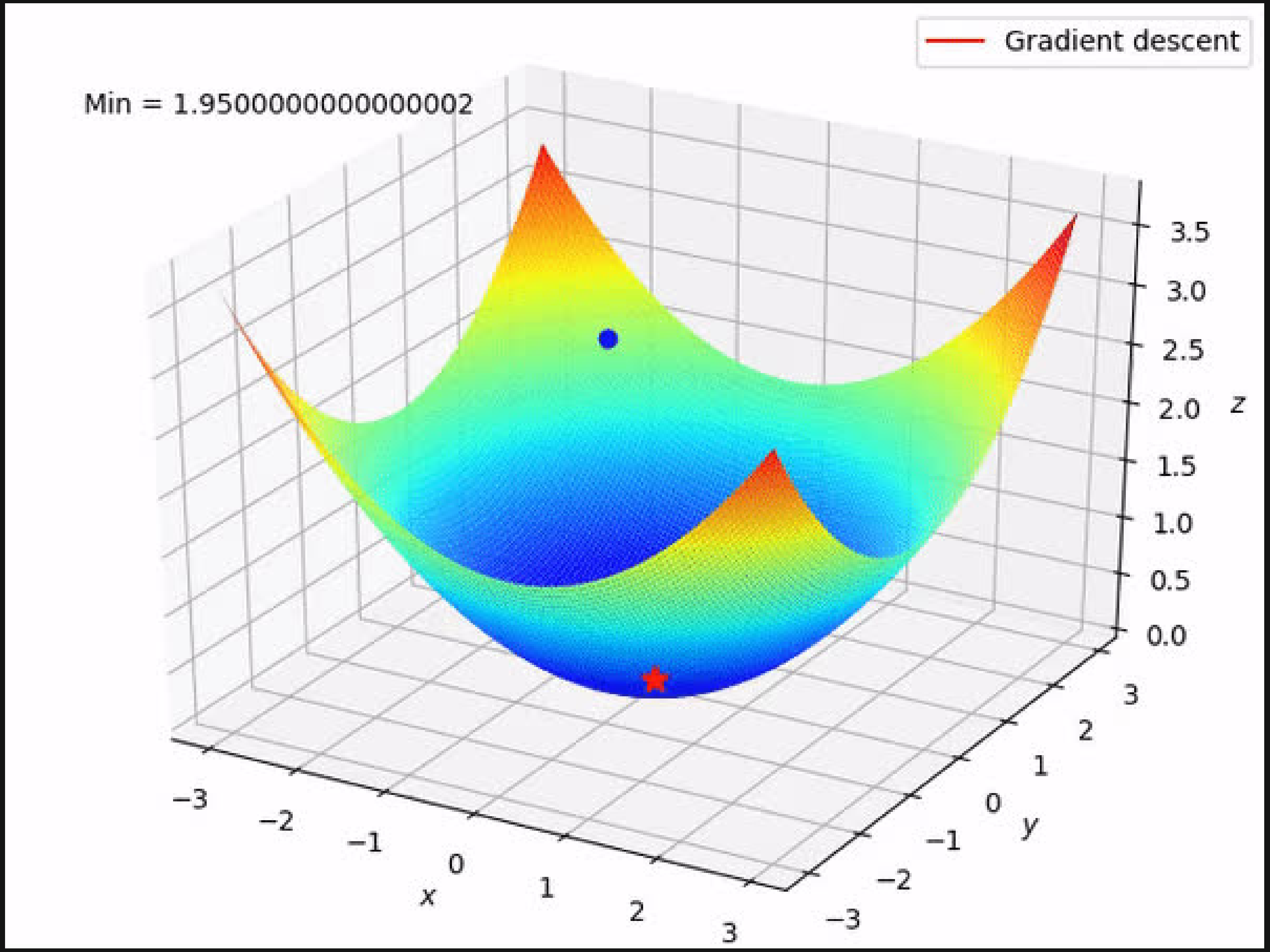
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

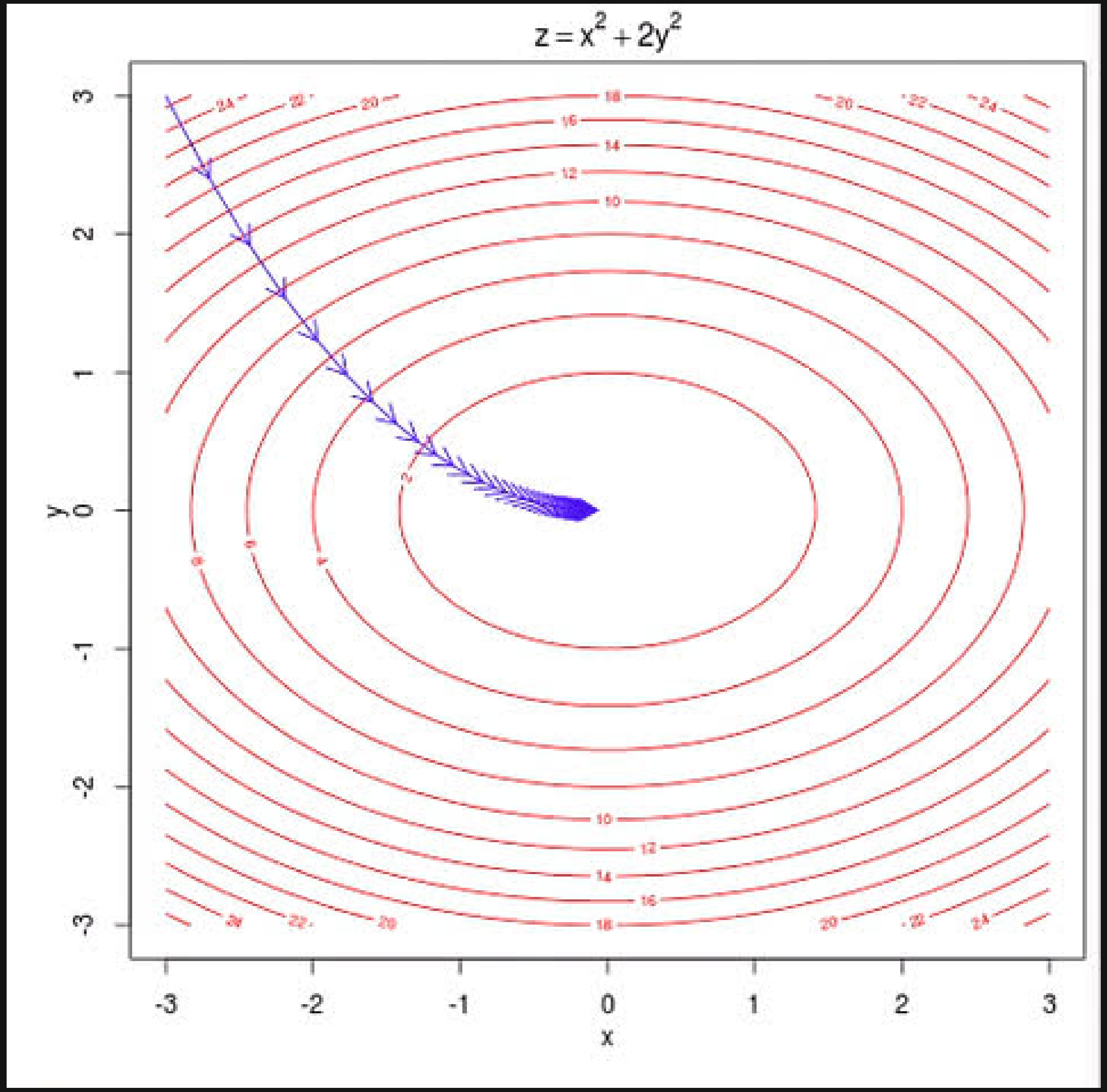
$$(\text{for } j = 1 \text{ and } j = 0)$$

This is for a cost function which depends on two parameters only

More on Gradient Descent

You can think of it as how a rolling ball would behave if it were let go on a surface like this. It would try to roll to the bottom of this "hill" - which is what we want!**



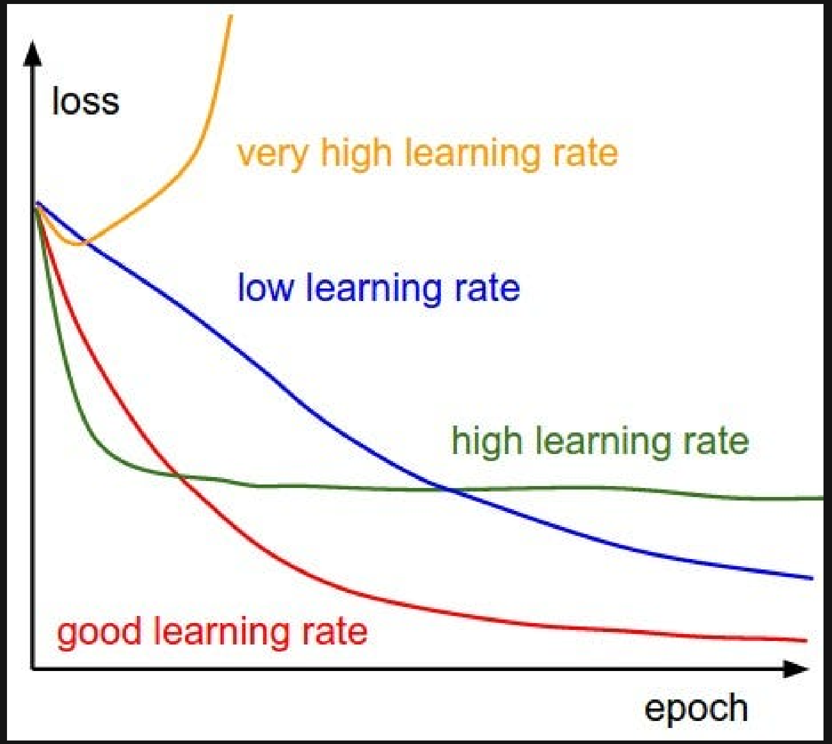


The Learning Rate α

The learning rate is a hyperparameter in deep learning algorithms that determines the step size or the rate at which the model learns from the training data during the optimization process. It controls how quickly or slowly the model parameters are updated during gradient descent or other optimization algorithms.

Small α

A small **α** will result in a very slow convergence to minima



Large α

A Large **α** might lead to overshooting the minima and the cost function actually starts to uncrease after every iteration!!

A good way to determine an optimum α is to plot the cost function with time . If it is increasing , learning rate should be reduced and if the convergence is very slow , then it can be increased to achieve quicker convergence

We'll move to

Code Implementation .