# Introduction

Deep learning is a branch of Machine learning and uses artificial neural networks to learn patterns in data. As a matter of fact, I feel Deep Learning is easy to comprehend initially than other machine learning algorithms such as Naïve Bayes, Random Forest, K-Nearest, Support Vector Machine, Gaussian Model etc.

Similar to machine learning, deep learning also has supervised, unsupervised and reinforcement learning.

But before we get ahead of ourselves, let us start from what we know about traditional programming.

# Slide 4 - Traditional Programming – Examples

We're used to creating applications by breaking down requirements into composable problems that can then be coded against.

For example, if we have to write an application that figures out a stock analytic, maybe the price divided by the earnings, we can usually write code to get the values from a data source, do the calculation and then return the result.

Or if we're writing a game we can usually figure out the rules. For example, if the ball hits the brick then the brick should vanish and the ball should rebound. But if the ball falls off the bottom of the screen then maybe the player loses their life.

# Slide 5 - Traditional Programming vs machine learning

We can represent that with this diagram. Rules and data go in answers come out. Rules are expressed in a programming language and data can come from a variety of sources from local variables all the way up to databases.

Machine learning rearranges this paradigm where we put answers & data in and then we get rules out. So instead of us as developers figuring out the rules when should the brick be removed, when should the player's life end, or what's the desired analytic for a stock, what we will do is we can get a bunch of examples for what we want to see and then have the computer figure out the rules.

# Slide 6 - Applicability of machine learning

Now, this is particularly valuable for problems that you can't solve by figuring the rules out for yourself. Consider this example of activity recognition.

If I'm building a device that detects if somebody is say walking and I have data about their speed, I might write code like this and if they're running well that's a faster speed so I could adapt my code to this and if they're biking, well that's not too bad either. I can adapt my code like this.

But then I have to do golf recognition too, now my concept becomes broken. But not only that, doing it by speed alone of course is quite naive. We walk and run at different speeds uphill and downhill and other people walk and run at different speeds to us. So, let's go back

to this diagram.

Ultimately machine learning is very similar but we're just flipping the axes. So instead of me trying to express the problem as rules when often that isn't even possible, I'll have to compromise. The new paradigm is that I get lots and lots of examples and then I have labels on those examples and I use the data to say this is what walking looks like, this is what running looks like, this is what biking looks like and yes, even this is what golfing looks like. So, then it becomes answers and data in with rules being inferred by the machine. A machine learning algorithm then figures out the specific patterns in each set of data that determines the distinctiveness of each. That's what's so powerful and exciting about this programming paradigm.

It's more than just a new way of doing the same old thing. It opens up new possibilities that were infeasible to do before. In the next few slides, I'm going to show you the basics of creating a neural network which is the workhorse of doing this type of pattern recognition.

A neural network is just a slightly more advanced implementation of machine learning and we call that deep learning. But fortunately, it's actually very easy to code. So, we're just going to jump straight into deep learning.

We'll start with a simple one and then we'll move on to one that does computer vision in about 10 lines of code. But let's start with a very simple "Hello World" example.

# Slide 8 - Can you figure out the formula?

Earlier I mentioned that machine learning is all about a computer learning the patterns that distinguish things. Like for activity recognition, it was the pattern of walking, running and biking

that can be learned from various sensors on a device.

To show how that works, let's take a look at a set of numbers and see if you can determine the pattern between them. Okay, here are the numbers. There's a formula that maps X to Y. Can you spot it? Take a moment.

Well, the answer is Y equals 2X minus 1.

Whenever you see a Y, it's twice the corresponding X minus 1. If you figured it out for yourself, well done, but how did you do that? How would you think you could figure this out? Maybe you can see that the Y increases by 2 every time the X increases by 1.

So it probably looks like Y equals 2X plus or minus something. Then when you saw X equals 0 and Y equals minus 1, so you thought hey that the something is a minus 1, so the answer might be Y equals 2X minus 1. You probably tried that out with a couple of other values and see that it fits.

Congratulations, you've just done the basics of machine learning in your head.

# Slide 11 - Understanding loss functions for neural nets

**Linear Regression**Linear regression is a very basic type of supervised machine learning, which tries to find the best possible straight line to describe main trend in underlying training data. Starting from some random position, it adjusts/moves the line until it finds the position that gives the minimum total average distance of data points from the line. It uses MEAN SQUARED ERROR (MSE) as a loss function and STOCHASTIC GRADIENT DESCENT (SGD) as an optimization algorithm.

Stochastic gradient descent optimization method knows how to move line into direction which will likely lower the error.

In its basic form, for one input x, and one output y, the model of this algorithm is basic mathematical formula for straight line y=k\*x+n and training comes down to figuring out parameters k and n. The same procedure can be applied to problems with multiple inputs.

**Logistic Regression**

Logistic regression is a basic binary (yes/no) classification algorithm, that works in a same way as linear regression, just instead of adjusting straight line, it adjusts so called SIGMOID function.

The output of a logistic regression is probability that given input belongs to some category (eg. spam/not spam, fraud/not fraud).

Although Mean Squared Error can be used as a loss function, good practice is to use another type of loss function called CROSS ENTROPY, which are better suited for classification problems.

The basic principle is same: it uses optimization method to find values for parameters of sigmoid function that give the minimum of the loss function (prediction error for the given data set).

**Convolutional Neural Network**

Convolutional neural networks are neural networks specialized for image recognition tasks.  You can think of them as a Feed Forward Networks with trainable image filters in front.

These image filters perform photoshop-like image filtering (in so called convolutional layers) and resizing (in pooling layers). These image filters (also known as kernels) are able to learn to extract, and recognize  patterns from input image.

Convolutional networks can be used for visual recognition tasks like image classification and object detection.

# Slide 12 - Building A single neuron Neural network

Okay, here is our first line of code. This is written using Python and TensorFlow and an API in TensorFlow called Keras.

Keras makes it really easy to define neural networks. A neural network is basically a set of functions which can learn patterns. Don't worry if there were a lot of new concepts here. They will become clear quite quickly as you work through them.

The simplest possible neural network is one that has only one neuron in it, and that's what this line of code does.

In Keras, you use the word dense to define a layer of connected neurons. There's only one dense here. So there's only one layer and there's only one unit in it, so it's a single neuron.

Successive layers are defined in sequence, hence the word sequential. But as I've said, there's only one. So you have a single neuron. You define the shape of what's input to the neural network in the first and in this case the only layer, and you can see that our input shape is super simple. It's just one value.

# Slide 13 - Building A single neuron Neural network – Loss & Optimizer

You've probably seen that for machine learning, you need to know and use a lot of math, calculus probability and the like.

The nice thing for now about TensorFlow and keras is that a lot of that math is implemented for you in functions.

There are two function roles that you should be aware and these are **loss functions** and **optimizers**.

I like to think about it this way. The neural network has no idea of the relationship between X and Y, so it makes a guess. Say it guesses Y equals 10X minus 10. It will then use the data that it knows about, that's the set of Xs and Ys that we've already seen to measure how good or how bad its guess was. The loss function measures this and then gives the data to the optimizer which figures out the next guess.

Then the logic is that each guess should be better than the one before. As the guesses get better and better, an accuracy approaches 100 percent, the term convergence is used.

In this case, the loss is mean squared error and the optimizer is SGD which stands for stochastic gradient descent.

# Slide 14. - Building A single neuron Neural network – Train

Our next step is to represent the known data. These are the Xs and the Ys that you saw earlier.

The np.array is using a Python library called numpy that makes data representation particularly much easier.

So here you can see we have one list for the Xs and another one for the Ys.

The training takes place in the fit command. Here we're asking the model to figure out how to fit the X values to the Y values.

The epochs equals 500 value means that it will go through the training loop 500 times. This training loop is what we described earlier.

Make a guess, measure how good or how bad the guesses with the loss function, then use the optimizer and the data to make another guess and repeat this.

When the model has finished training, it will then give you back values using the predict method.

So it hasn't previously seen 10, and what do you think it will return when you pass it a 10? Now you might think it would return 19 because after all Y equals 2X minus 1, and you think it should be 19.

But when you try this in the workbook yourself, you'll see that it will return a value very close to 19 but not exactly 19. Now why do you think that would be?

Ultimately there are two main reasons. The first is that you trained it using very little data. There's only six points. Those six points are linear but there's no guarantee that for every X, the relationship will be Y equals 2X minus 1. There's a very high probability that Y equals 19 for X equals 10, but the neural network isn't positive. So it will figure out a realistic value for Y. That's the second main reason. When using neural networks, as they try to figure out the answers for everything, they deal in probability.