STUDY ORIENTED PROJECT REPORT

DEEP LEARNING FOR OPTIONS PRICING

BY

Name of the Student	ID Number
Aiswarya Subramanian	***2015B4A70585G***

Prepared on completion of the
SOP Course No. CS F326

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

April, 2018

ACKNOWLEDGMENTS

The work described in this dissertation was carried out in BITS Pilani, Goa Campus, Department of Computer Science, from 1st of January, 2019 to 20th of April, 2019.

I am grateful especially to my supervisor, Dr. Partha Saha for his valuable comments and good advice. I am also very thankful to my family and friends, who supported and motivated me throughout my university studies.

ABSTRACT

Financial markets offer high amount of incentive for correctly forecasting different asset classes. Due to the inherently noisy and non-linear nature of market indices, drawing a forecast of the markets' behaviour becomes a challenging task. The Black & Scholes formula for theoretical pricing of options exhibits certain systematic biases, as observed prices in the market differs from the formula. Amongst nonparametric approaches used to improve accuracy of the model, Artificial Neural Networks are found as a promising alternative.

This project examined the application of neural networks in options pricing.

CONTENTS

INTRODUCTION

Parallel with the increasing importance of derivatives in the world of financial markets,many pricing techniques have been developed in order to meet the need of estimation of true value. Futures and Options are widely used by investors and traders to leverage their investments to gain multi-fold returns and/or manage their risk exposure.

The stock markets have also significantly developed with many new asset pricing models for various financial instruments. In 1973, Black and Scholes published their famous formula for European-style option pricing which solves a parabolic partial differential equation with a final condition based on continuously revised delta hedging.

A new approach was taken by Hutchinson et al. (1994) who used an artificial neural network model which was able to naturally explain the option price without assumptions made by parametric models (i.e. Black-Scholes formula).

The project aims to verify the following hypothesis that:

Hypothesis : Artificial neural network approach performs better in options price prediction than conventional parametric Black-Scholes formula.

LITERATURE REVIEW

OPTIONS :

Definition (Wikipedia) : In finance, an **option** is a contract which gives the buyer (the owner or holder of the option) the right, but not the obligation, to buy or sell an underlying asset or instrument at a specified strike price prior to or on a specified date, depending on the form of the option.

There are two kinds of options: (learnt from this video : https://www.youtube.com/watch?v=EfmTWu2yn5Q&pbjreload=10)

1. Call option

2. Put Option

Explanation with an example:

Call Option :

Call options are an agreement that give the option buyer the right, but not the obligation, to buy a stock, bond, commodity or other instrument at a specified price within a specific time period. The stock, bond, or commodity is called the underlying asset.

For example:

There is ‘A’ who wants to sell a share (say) at Rs. 100 per share. ‘B’ wants to buy this share at this striking price of Rs. 100 per share, but at the moment, he doesn’t have the money to purchase 100 of these shares. So he buys a call option, at (say) Rs. 250, and this is paid to ‘A’, and they fix the expiry for the call option (say) as 3 months. This means that, ‘A’ will not sell the share for the next three months, and whenever ‘B’ has money, he can buy all 100 shares at the striking price of Rs. 100 per share.

Here, three things can happen:

1. The price of the share increased to Rs.110 per share, before the 3 months expired. In this case, ‘B’ shall be happy as, he can still buy the share at Rs. 100 per share. He just had to pay an extra 250 for the option. On the other hand, ‘A’ loses Rs. 10 on each share, he only got Rs. 250 which ‘B’ had paid earlier.
2. The price of the share decreased to Rs.90 per share, before the 3 months expired. In this case, ‘A’ shall be happy as, he can still sell the share at Rs. 100 per share. On the other hand, ‘B’ loses Rs. 10 on each share, if he chooses to exercise his option. So in this case, ‘B’ can choose to not buy the share and walk away, only losing Rs. 250 which he paid earlier.
3. The price remains the same. In this case also, ‘B’ can choose to buy the shares at Rs. 100 for each share, or to not buy the share and walk away, only losing Rs. 250 which he paid earlier.

If the option expires, ‘A’ has no right over the shares anymore and he lost R. 250 for the option.

Put Option:

A put option is an option contract giving the owner the right, but not the obligation, to sell a specified amount of an underlying security at a specified price within a specified time frame.

For example:

‘A’ has a truck worth (say) Rs. 40000, but he is scared that he might meet with an accident or it might get stolen. So, he approaches ‘B’ for security insurance. He pays (say) Rs. 1500 as security amount to ‘B’ and secures his truck, with an expiry date of one year (say).

Here again three things can happen:

1. The truck does not meet with any accident nor gets stolen in the next one year, in which case, ‘A’ loses his 1500 and ‘B’ is happy to gain a profit.
2. The truck gets damaged before an year gets over, and ‘A’ needs say Rs. 10000 for repair. So he claims his insurance, in which case, ‘B’ pays Rs.

10000 for the repair, and ‘A’ is happy to have paid only Rs. 1500 for security.

3. The truck gets stolen. Again, ‘A’ claims his insurance and ‘B’ pays the full Rs. 40000 for buying a new truck. Here again, ‘A’ is happy for getting paid for a new truck.

   Here again, if option expires, ‘A’ loses his Rs. 1500.

   So in this case, ‘B’ might check the driving record of ‘A’. If he sees that the driver is poor at driving, he might as well keep the option price at a higher price so that he does not incur huge losses. But at the same time, if the driver has excellent driving records, ‘B’ may keep the price of an option lesser than this value.

Options may also be classified according to their exercise time:

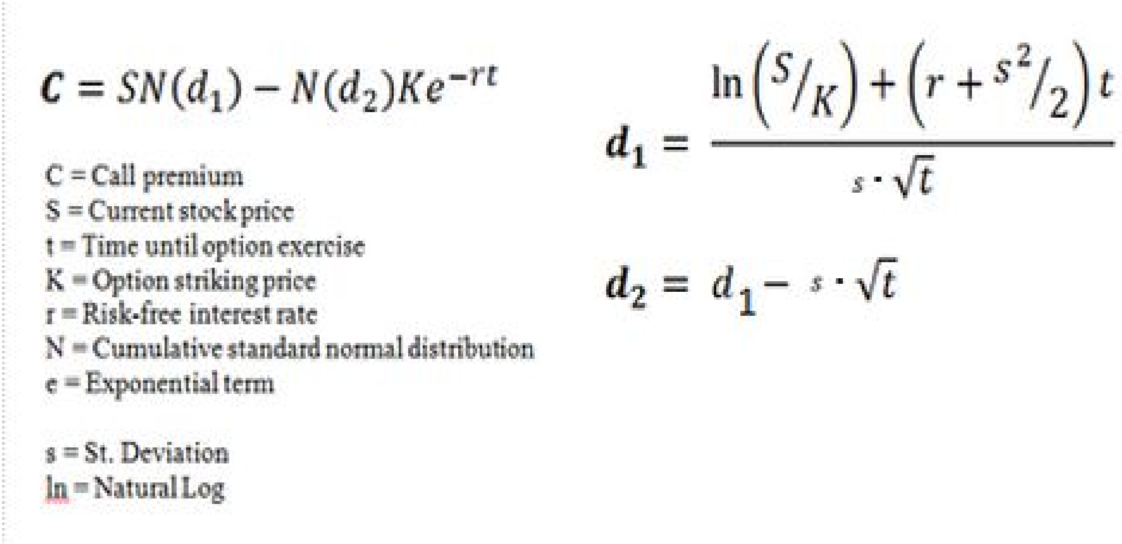
- **European style options** may be exercised only at the expiration date.
- **American style options** can be exercised anytime between purchase and expiration date.

The above mentioned classification of options is extremely important because choosing between European-style or American-style options will affect our choice for the option pricing model.

Options Pricing Models:

Option Pricing Models are mathematical models that use certain variables to calculate the theoretical value of an option. The theoretical value of an option is an estimate of what an option should worth using all known inputs.

BLACK SCHOLES FORMULA FOR OPTIONS PRICE CALCULATION



$$C = SN(d_1) - N(d_2)Ke^{-rt}$$

C = Call premium
S = Current stock price
t = Time until option exercise
K = Option striking price
r = Risk-free interest rate
N = Cumulative standard normal distribution
e = Exponential term

s = St. Deviation
ln = Natural Log

$$d_1 = \frac{\ln\left({}^{S}/_{K}\right) + \left(r + {}^{s^2}/_{2}\right)t}{s \cdot \sqrt{t}}$$

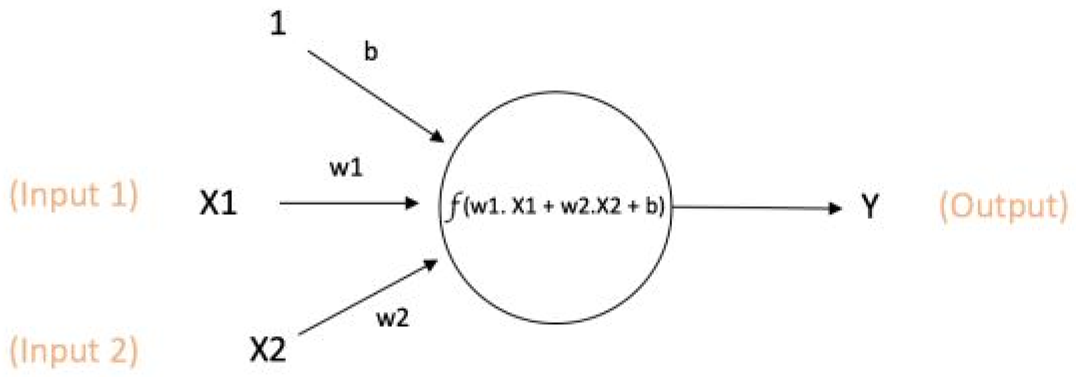
$$d_2 = d_1 - s \cdot \sqrt{t}$$

Image Courtesy: Investopedia

NEURAL NETWORKS :

Before we go into neural networks, let’s get a basic idea of the simplest form of neural networks, that is the perceptron.

The perceptron model is one of the pioneers of Machine Learning algorithms available today.
It is a simple architecture which falls under the category of supervised learning models. The basic unit of this architecture is a perceptron neuron. The neuron receives input from other neurons or receives input from an external source and then calculates the output. Each input is complemented with “weight” (w), the weight of which depends on the relative importance of the other inputs. The neuron applies the function f (defined as follows) to the weighted input sum, as shown:

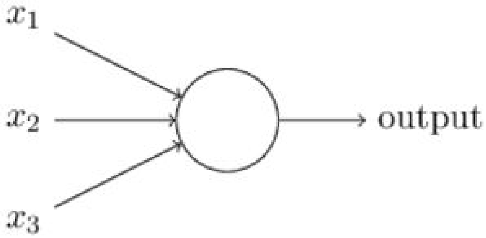


Output of neuron $= Y = f(w1.X1 + w2.X2 + b)$

This network accepts the numeric inputs of X1 and X2 with weights of w1 and w2, respectively. There is also input 1 with weight b (called “bias”).

Weights(source):

To understand more about weights, let’s take the following example:



There are three inputs x1, x2 and x3.

In order to compute an output, here’s what you could do (O = Output):

a) Sum up the inputs. O = x1 + x2 + x3

b) Apply a linear mathematical operation to the inputs. O = (x1 * x2) + x3

To understand the best possible approach to this problem, we need to realize that since x1,x2 and x3 are binary inputs here, multiplying one or more inputs together can result in a 0 output if one of the inputs is 0. So, this tells us that approach (b) can have its output as O = x3, if x1 or x2 or both are equal to zero. If only one of them is equal to 0, this can lead to a loss of information as the input value gets multiplied by 0.

Now, let us see the simple approach (a), understand why it’s partially correct and what we can do to make it fully correct. Approach (a) sure takes into account all the inputs without one being immediately affected by the other. However, it does not take in the relative importance of the inputs.

Consider a simple example where you want to make a perceptron for predicting whether it will rain today or not. You have a binary output as O, which can take values of 0 or 1 and inputs x1 and x2.

Let x1 be 1 if the weather is humid today, 0 if the opposite. Let x2 be 1 if you are wearing a red shirt today and 0 if not. We can see here that wearing a red shirt has almost no correlation with the possibility of rainfall.

So, a possible output function can be:

O = x1 + **0.1** * x2

Here’s what the factor **0.1** does. If x2 is 0, 0.1*x2 is still 0, but if x2 is 1, 0.1*x2 will be 0.1, not 1. It basically brings down the importance of the input x2 from 1 to 0.1, and hence, it is called the **‘weight’** of the input x2.

Consider x2 to be 1 for now. O will still be (0 + 0.1) = 0.1 or (1 + 0.1) = 1.1, i.e. not a binary value. In order to make it one, what we can do is use a ‘**threshold’** for the total value of O. We can say, like, O is 1 if (x1 + 0.1 * x2) > 1

and O = 0 if (x1 + 0.1 * x2) < 1. We have thus solved our problem

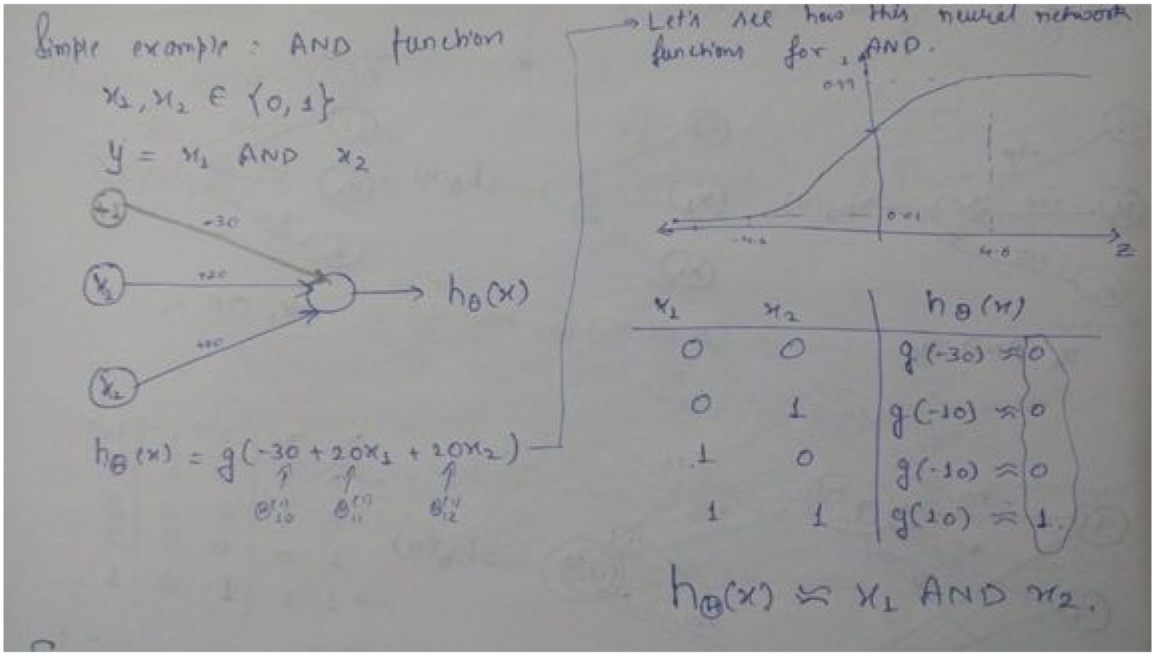
In general:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Bias(source):

In simplest words, it is that output of the neural net when it has absolutely zero input. Let us see with an example:

When you are implementing an AND gate with a simple perceptron (effectively, a single neuron), you take into account both the inputs, their corresponding weights, and try to predict the output, like so:



Simple example : AND function

$x_1, x_2 \in \{0, 1\}$

$y = x_1$ AND x_2

$h_\theta(x) = g(\underset{\overset{\uparrow}{\theta^{(1)}_{10}}}{-30} + \underset{\overset{\uparrow}{\theta^{(1)}_{11}}}{20}x_1 + \underset{\overset{\uparrow}{\theta^{(1)}_{12}}}{20}x_2)$

Let's see how this neural network functions for AND.

x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1.$

$h_\theta(x) \approx x_1$ AND x_2.

So you see, everything fell into place when we took bias into account. Had bias not been there, the optimal weights on the two inputs (so that we'd have the correct output) would have been extremely computationally expensive to be determined, if not at all impossible. By including bias, and by giving it an appropriate value, we could effectively and (more importantly) simply, map all the four possible combination of inputs to their outputs.

In effect, **a bias value allows you to shift the activation function to the left or right**, which may be critical for successful learning.

Activation Function(source):

The function f is nonlinear and is called the Activation Function. The function of the activation function is to introduce nonlinearity into the output of the neuron. Because most of the real world data is non-linear, we hope that neurons can learn non-linear functional representations, so this application is crucial.

The Activation Functions can be basically divided into 2 types-

1. Linear Activation Function
2. Non-linear Activation Functions

1. Linear Activation Function:

Equation : f(x) = x

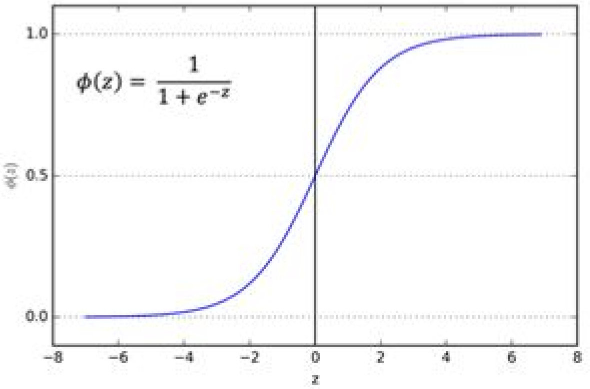
Range : (-infinity to infinity)

It doesn’t help with the complexity or various parameters of usual data that is fed to the neural networks.

2. Non - Linear Activation Function:

1. Sigmoid or Logistic Activation Function

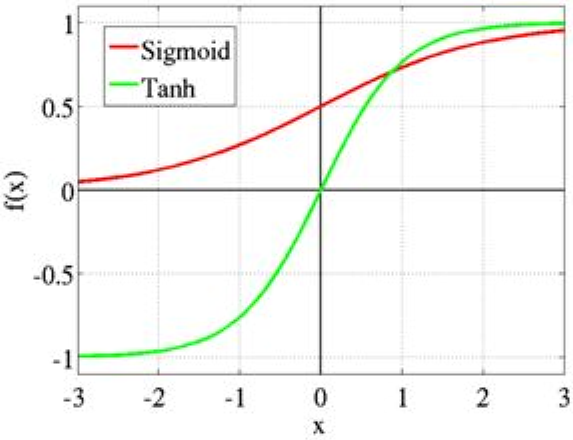
The Sigmoid Function curve looks like a S-shape.



The main reason why we use sigmoid function is because it’s range is between **0 and 1.** Therefore, it is especially used for models where we have to **predict the probability** as an output.
The function is **differentiable**. That means, we can find the slope of the sigmoid curve at any two points.
The function is **monotonic** but function’s derivative is not.
The logistic sigmoid function can cause a neural network to get stuck at the training time.
The **softmax function** is a more generalized logistic activation function which is used for multiclass classification.

2. Tanh or hyperbolic tangent Activation Function

The tanh function is also like logistic sigmoid but the range is from -1 to 1.The tanh is also sigmoidal (s - shaped).

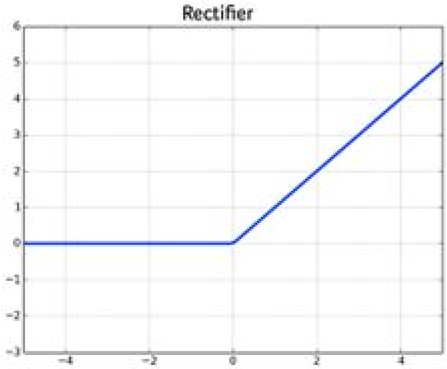


The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.
The function is **differentiable**.
The function is **monotonic** while its **derivative is not monotonic**.
The tanh function is mainly used classification between two classes.
Both tanh and logistic sigmoid activation functions are used in feed-forward nets.

3. ReLU (Rectified Linear Unit) Activation Function

The ReLU is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning.

ReLU (Rectified Linear Activation)



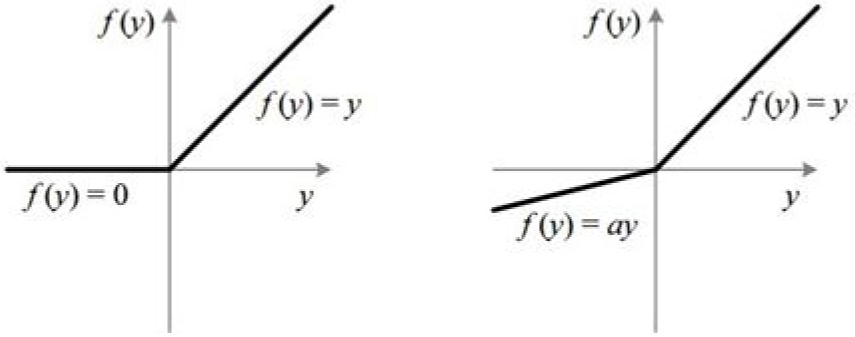
$$RELU(x)=\begin{cases}0 \ if \ x<0\\ x \ if \ x>=0\end{cases}$$

As you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

Range: [0 to infinity)

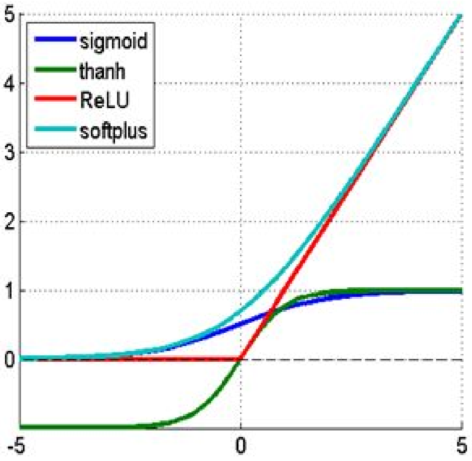
The function and its derivative **both are monotonic**.

The issue is that this function maps all negative values to zero. To overcome this problem, “Leaky ReLU” has been designed, which is as follows:



(a) ReLU (b) Leaky ReLU

The graphs of all the activation functions are shown below:



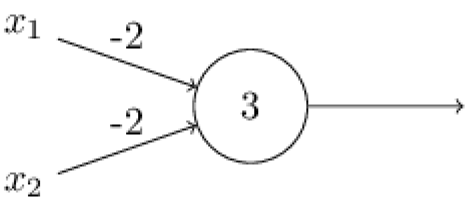
Working of the Perceptron:

As already seen above, the main component of the model is the neuron which takes in various inputs and gives a single output. The neuron's output, 0 or 1, is determined by whether the weighted sum ∑jwjxj is less than or greater than some *threshold value*, which is the bias.

A way you can think about the perceptron is that it's a device that makes decisions by weighing up evidence. If one of the weights is greater than all the other weights, that means, the feature or the input corresponding to that weight is of the highest priority. Finally the threshold or the bias value states the minimum value for the input which is needed for the neuron to fire. Lower value of bias indicates that the decision is likely to be accepted. For a perceptron with a really big bias, it's extremely easy for the perceptron to output a 1. But if the bias is very negative, then it's difficult for the perceptron to output a 1.

Now other than making decisions, perceptrons can also be used to implement logical functions.

For example, suppose we have a perceptron with two inputs, each with weight −2−2, and an overall bias of 3. Here's our perceptron:



Then we see that input 0,0 produces output 1, since (−2)∗0+(−2)∗0+3=3(−2)∗0+(−2)∗0+3=3 is positive. Similar calculations show that the inputs 0,1 and 1,0 produce output 1. But the input 1,1 produces output 0, since (−2)∗1+(−2)∗1+3=−1(−2)∗1+(−2)∗1+3=−1 is negative. And so our perceptron implements a NAND gate!
And because NAND gates are universal for computation, it follows that perceptrons are also universal for computation.

A Sigmoid Neuron:

In case of a perceptron neuron, the activation function is hard assignment, as shown :

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

So when we are training, for example, we are doing handwritten recognition. And we'd like the network to learn weights and biases so that the output from the network correctly classifies the digit. So, to see the learning, we may change the weights and bias slightly.What we'd like is for this small change in weight to cause only a small corresponding change in the output from the network.

For example, if **‘1’** was misclassified as **‘7’,** we will keep changing the weights slowly so that it correctly classifies. The problem is that this isn't what happens when our network contains perceptrons. In fact, a small change in the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to completely flip, say from 0 to 1. That flip may then cause the behaviour of the rest of the network to completely change in some very complicated way.

We overcome this problem by introducing a new type of artificial neuron called a ***sigmoid*** neuron. Sigmoid neurons are similar to perceptrons, but modified so that small changes in their weights and bias cause only a small change in their output. That's the crucial fact which will allow a network of sigmoid neurons to learn.

In a sigmoid neuron, unlike a perceptron neuron, where inputs can be only either 0 or 1, inputs can be any value in between 0 and 1 (including 0 and 1). The activation function used is the sigmoid function.
To put it all a little more explicitly, the output of a sigmoid neuron with inputs x1,x2,...,..., weights w1,w2,......, and bias b is:

$$1/(1+\exp(-\sum_j w_j x_j - b))$$

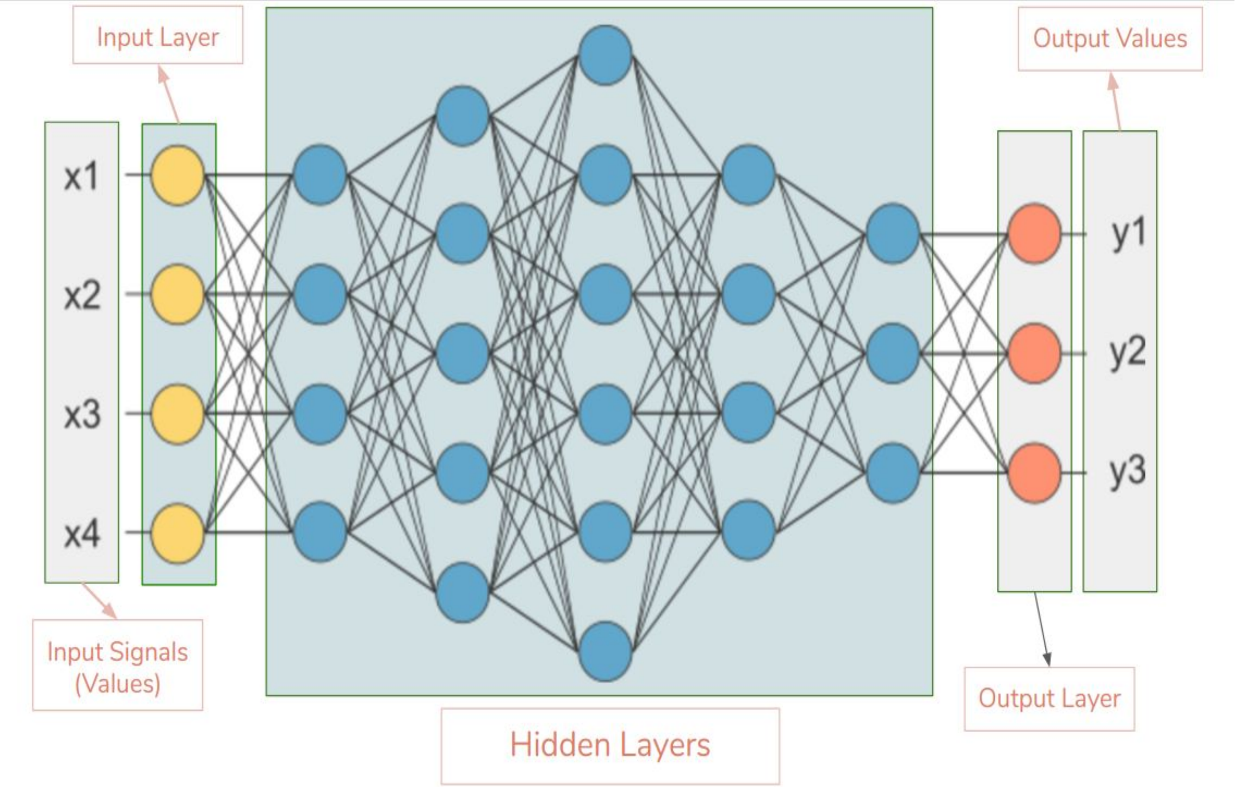
To understand the similarity to the perceptron model, suppose $z \equiv w \cdot x + b$ is a large positive number. Then $e^{-z} \approx 0$ and so $\sigma(z) \approx 1$. In other words, when $z = w \cdot x + b$ is large and positive, the output from the sigmoid neuron is approximately 1, just as it would have been for a perceptron. Suppose on the other hand that $z = w \cdot x + b$ is very negative. Then $e^{-z} \to \infty$, and $\sigma(z) \approx 0$. So when $z = w \cdot x + b$ is very negative, the behaviour of a sigmoid neuron also closely approximates a perceptron. It's only when $w \cdot x + b$ is of modest size that there's much deviation from the perceptron model.

So if our sigmoid function became the step function, the our sigmoid neuron becomes perceptron neuron. So the sigmoid neuron is smoothed version of perceptron neuron.

The smoothness of σ means that small changes Δw_j in the weights and Δb in the bias will produce a small change Δoutput in the output from the neuron.

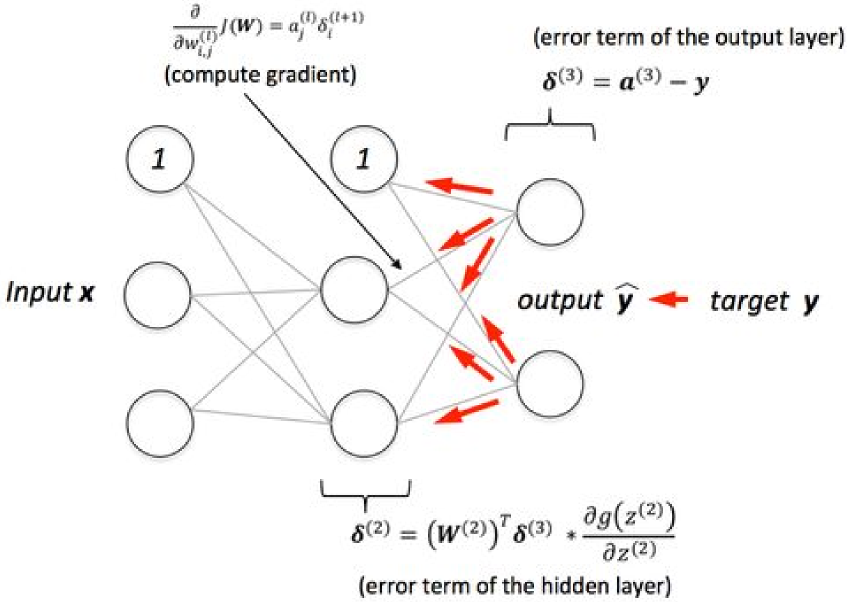
Artificial Neural Networks

***Artificial neural networks** (**ANNs**) or **connectionist systems** are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve performance on) tasks by considering examples, generally without task-specific programming —Wikipedia*



Forward Propagation — Forward propagation is a process of feeding input values to the neural network and getting an output which we call predicted value. Sometimes we refer forward propagation as inference. When we feed the input values to the neural network's first layer, it goes without any operations. Second layer takes values from first layer and applies multiplication, addition and activation operations and passes this value to the next layer. Same process repeats for subsequent layers and finally we get an output value from the last layer.

Back Propagation -


$\frac{\partial}{\partial w_{i,j}^{(l)}} J(W) = a_j^{(l)} \delta_i^{(l+1)}$
(compute gradient)
(error term of the output layer)
$\delta^{(3)} = a^{(3)} - y$
1
1
Input **x**
output $\hat{y}$
target **y**
$\delta^{(2)} = (W^{(2)})^T \delta^{(3)} * \frac{\partial g(z^{(2)})}{\partial z^{(2)}}$
(error term of the hidden layer)

RECURRENT NEURAL NETWORKS:

*A **recurrent neural network** (**RNN**) is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.* -- Wikipedia

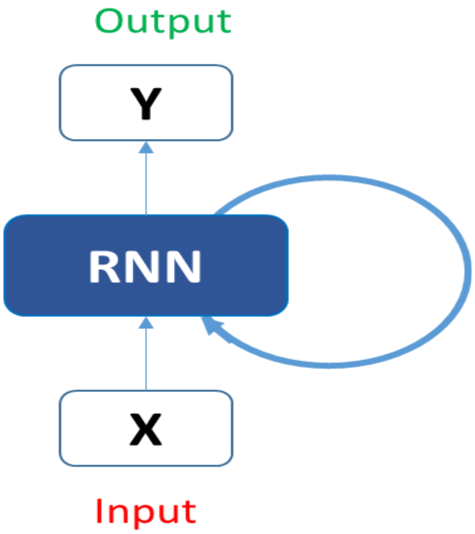


Image courtesy : Wikipedia

In a RNN, the information cycles through a loop. When it makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously. A usual RNN has a short-term memory.

A Recurrent Neural Network has two inputs, the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.

Also note that while Feed-Forward Neural Networks map one input to one output, RNN's can map one to many, many to many (translation) and many to one (classifying a voice).

Furthermore they also tweak their weights for both through gradient descent and Backpropagation Through Time. Backpropagation Through Time (BPTT) is basically just a fancy buzz word for doing Backpropagation on an unrolled Recurrent Neural Network.

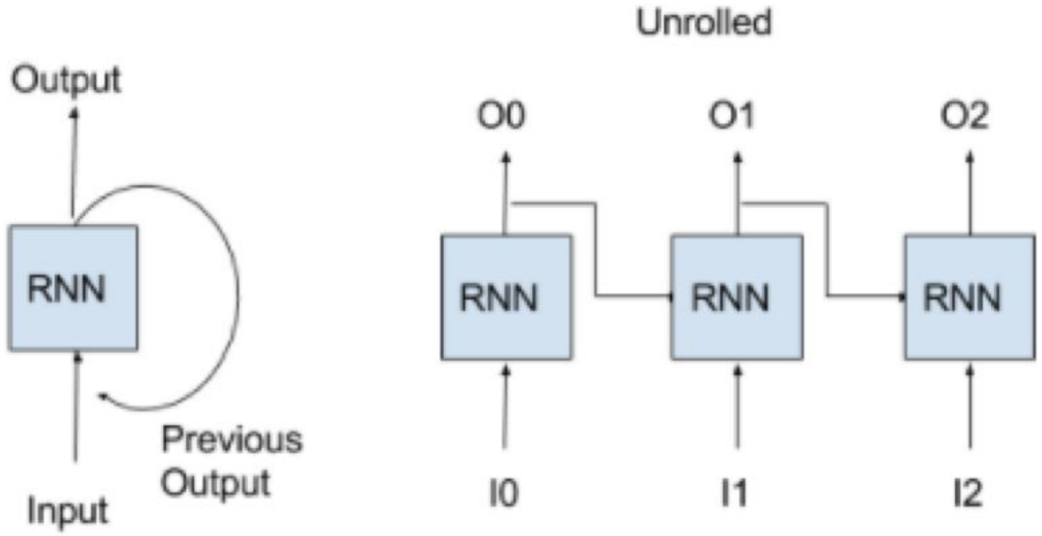


Image courtesy : Wikipedia

LSTMs - **Long-Short Term Memory**

One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. In cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information.

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.

In an LSTM you have three gates: input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn’t important (forget gate) or to let it impact the output at the current time step (output gate).

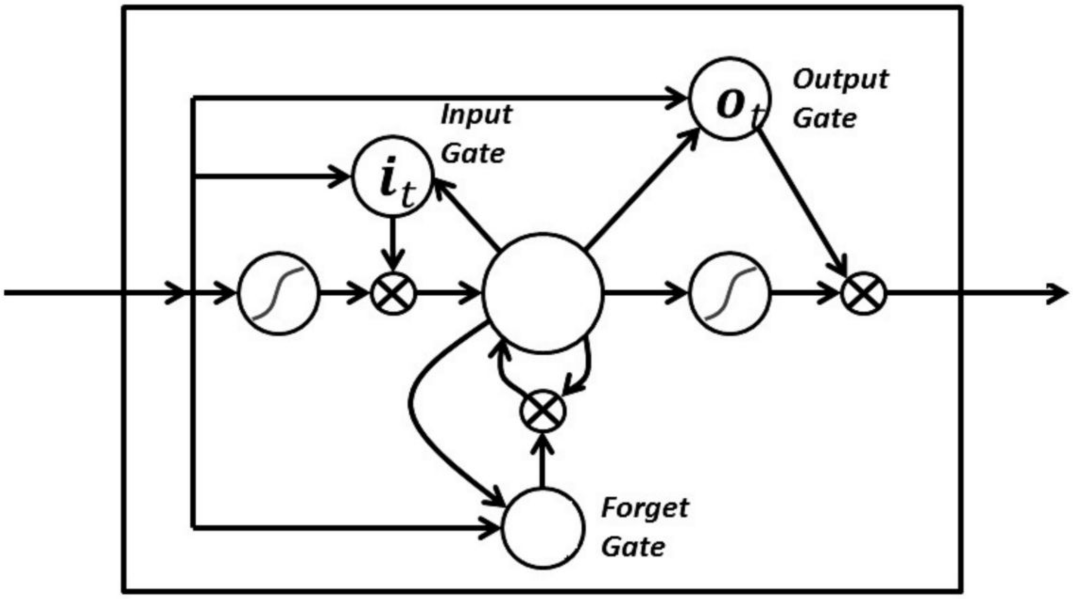


Image Courtesy : Medium

An Idea (taken from Analytics Vidya) :

The functioning of LSTM can be visualized by understanding the functioning of a news channel's team covering a murder story. Now, a news story is built around facts, evidence and statements of many people. Whenever a new event occurs you take either of the three steps.

Let's say, we were assuming that the murder was done by 'poisoning' the victim, but the autopsy report that just came in said that the cause of death was 'an impact on the head'. Being a part of this news team what do you do? You immediately **forget** the previous cause of death and all stories that were woven around this fact.

What, if an entirely new suspect is introduced into the picture. A person who had grudges with the victim and could be the murderer? You **input** this information into your news feed, right?

Now all these broken pieces of information cannot be served on mainstream media. So, after a certain time interval, you need to summarize this information and **output** the relevant things to your audience. Maybe in the form of "*XYZ turns out to be the prime suspect.*"

DESCRIPTION OF DATA

PART I : Predicting Stock Price Movement

Our dataset, which is stored in the .csv file named ‘RELIANCE.NS.csv’. The csv file contains daily OHLC data for the stock of Reliance trading on NSE for the time period from 1st January 1996 to 10th April 2019.
We then prepare the various input features which will be used by the artificial neural network to train itself for making the predictions. We define the following input features:

- High minus Low price
- Close minus Open price
- Three day moving average
- Ten day moving average
- 30 day moving average
- Standard deviation for a period of 5 days
- Relative Strength Index
- Williams %R

We then define the output value as price rise, which is a binary variable storing 1 when the closing price of tomorrow is greater than the closing price of today.

Some definitions: (from Wikipedia)

Moving Average : In statistics, a **moving average** (**rolling average** or **running average**) is a calculation to analyze data points by creating a series of averages of different subsets of the full data set.

Williams %R : **Williams %R**, or just **%R**, is a technical analysis oscillator showing the current closing price in relation to the high and low of the past N days (for a given N). It is calculated as

$$\%R = \frac{high_{Ndays} - close_{today}}{high_{Ndays} - low_{Ndays}} \times -100$$

Relative Strength Index : The relative strength index (RSI) is a technical indicator used in the analysis of financial markets. It is intended to chart the current and historical strength or weakness of a stock or market based on the closing prices of a recent trading period.

PART II : Options pricing

Training : We generate data as per black scholes formula and train the network. We are using more than 1,00,000 examples for the training purpose. 80% of the data is fed as training data and rest 20% is fed as validation data.

The dataset test.csv consists of the following fields namely, volatility, strike price, time and the current options price. There are 1530 data points. The data is taken from Scientific consultants. This data was used to test the model to analyse its performance.

ARCHITECTURE & RESULTS

I tried two models, first one was Artificial Neural Networks, and the second one was with LSTMs.

MODEL 1 : ARTIFICIAL NEURAL NETWORKS

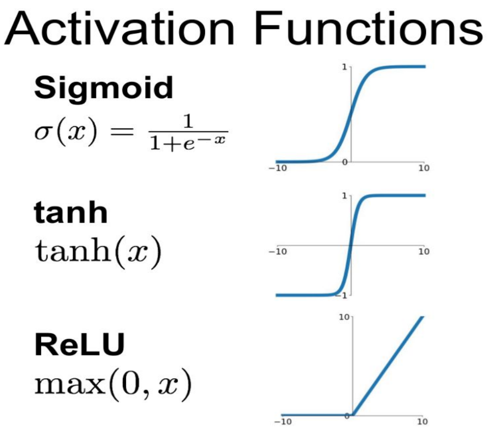
In this model, I had tried different architectures.

A. A 3-layer network having 3 input neurons (one for volatility, one for time remaining, and one for strike), 20 middle-layer or "hidden" neurons, and one output neuron (for theoretical option price).
B. A 3-layer network with 3 input neurons, 100 second-layer neurons, 50 third-layer neurons, and 1 output neuron.
C. A 4-layer network with 3 input neurons, 30 neurons in first layer, 60 in next two layers and 1 output neuron
D. A 5-layer network with 3 input neurons, 30 neurons in first layer, 60 in next two layers, 30 in next layer and 1 output neuron
E. A 5-layer network with 3 input neurons, 50 neurons in first layer, 100 in next two layers, 50 in next layer and 1 output neuron

In all cases, the input dimension = 3 (one for volatility, one for time remaining, and one for strike) and one output neuron. The basic function of the neural network is to predict the options price, given the above 3 inputs. Hence the output domain is the set of all positive real numbers. The given neural network architecture, shall represent an approximate function which try to fit the data.

Activation Function used : In both the architectures, as the output is all positive real numbers, our activation function at the output layer is the linear function. The final layer of the neural network will have one neuron and the value it returns is a continuous numerical value.

For the hidden layers, I tried three different activation functions : sigmoid, tanh & RELU.



Activation Functions

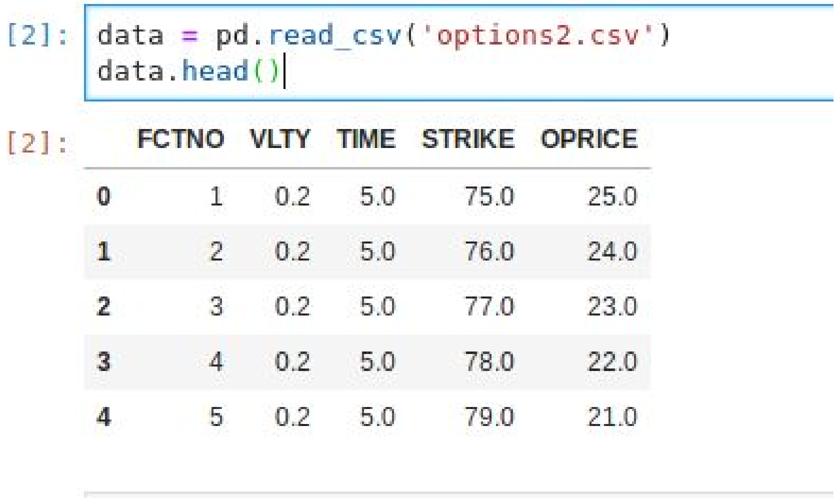
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$

tanh
$\tanh(x)$

ReLU
$\max(0, x)$

(image courtesy : google)

Data Preprocessing : Here, as our training data has features that are of different scales.



```
[2]: data = pd.read_csv('options2.csv')
     data.head()
```

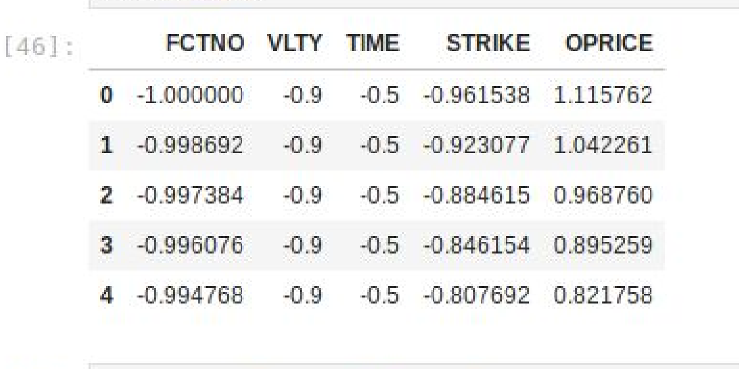
[2]:

	FCTNO	VLTY	TIME	STRIKE	OPRICE
0	1	0.2	5.0	75.0	25.0
1	2	0.2	5.0	76.0	24.0
2	3	0.2	5.0	77.0	23.0
3	4	0.2	5.0	78.0	22.0
4	5	0.2	5.0	79.0	21.0

We can clearly see that the ‘VLTY’ are relatively very small numbers when compared to ‘STRIKE’. Hence we need to do something known as “Feature Scaling”.

Feature Scaling or Standardization*: It is a step of Data Pre Processing which is applied to independent variables or features of data. It basically helps to normalise the data within a particular range.* -- GeeksforGeeks

Once we have scaled our variables, it looks like :

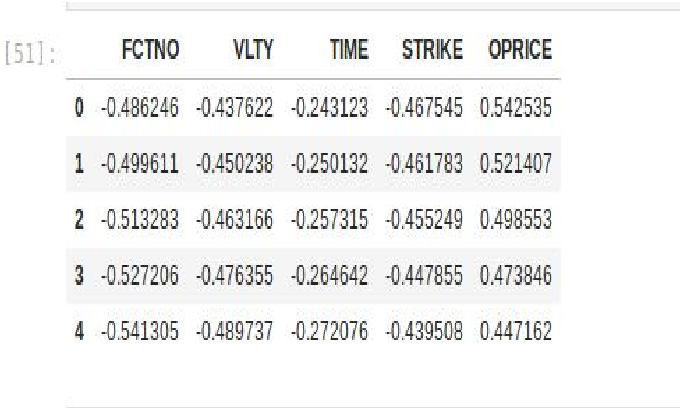


[46]:

	FCTNO	VLTY	TIME	STRIKE	OPRICE
0	-1.000000	-0.9	-0.5	-0.961538	1.115762
1	-0.998692	-0.9	-0.5	-0.923077	1.042261
2	-0.997384	-0.9	-0.5	-0.884615	0.968760
3	-0.996076	-0.9	-0.5	-0.846154	0.895259
4	-0.994768	-0.9	-0.5	-0.807692	0.821758

Next we try to normalise the data. Normalization is an example of preprocessing data to remove or reduce the burden from machine learning (ML) to learn certain *invariants*, that is, things which make no difference in the meaning of the symbol, but only change the representation.

Once we normalise the data, it looks like :



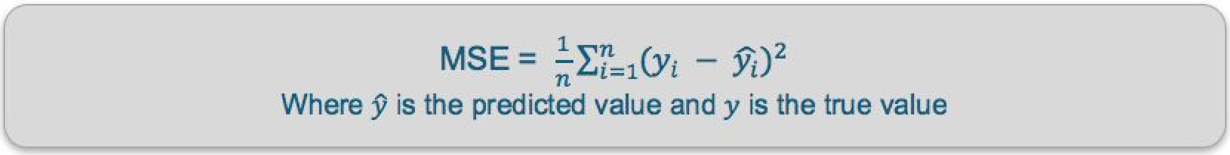
[51]:

	FCTNO	VLTY	TIME	STRIKE	OPRICE
0	-0.486246	-0.437622	-0.243123	-0.467545	0.542535
1	-0.499611	-0.450238	-0.250132	-0.461783	0.521407
2	-0.513283	-0.463166	-0.257315	-0.455249	0.498553
3	-0.527206	-0.476355	-0.264642	-0.447855	0.473846
4	-0.541305	-0.489737	-0.272076	-0.439508	0.447162

Now, the preprocessing of the data has been done, and can be used for training the model. As per the above architecture, using MSE as loss function, we train the model.

Loss Function : To understand the accuracy of the prediction, it is compared with the true value which is also a continuous number. As a regression type of problem, we use

Mean Squared Error : this finds the average squared difference between the predicted value and the true value.



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

Where $\hat{y}$ is the predicted value and y is the true value

Image Courtesy : Medium

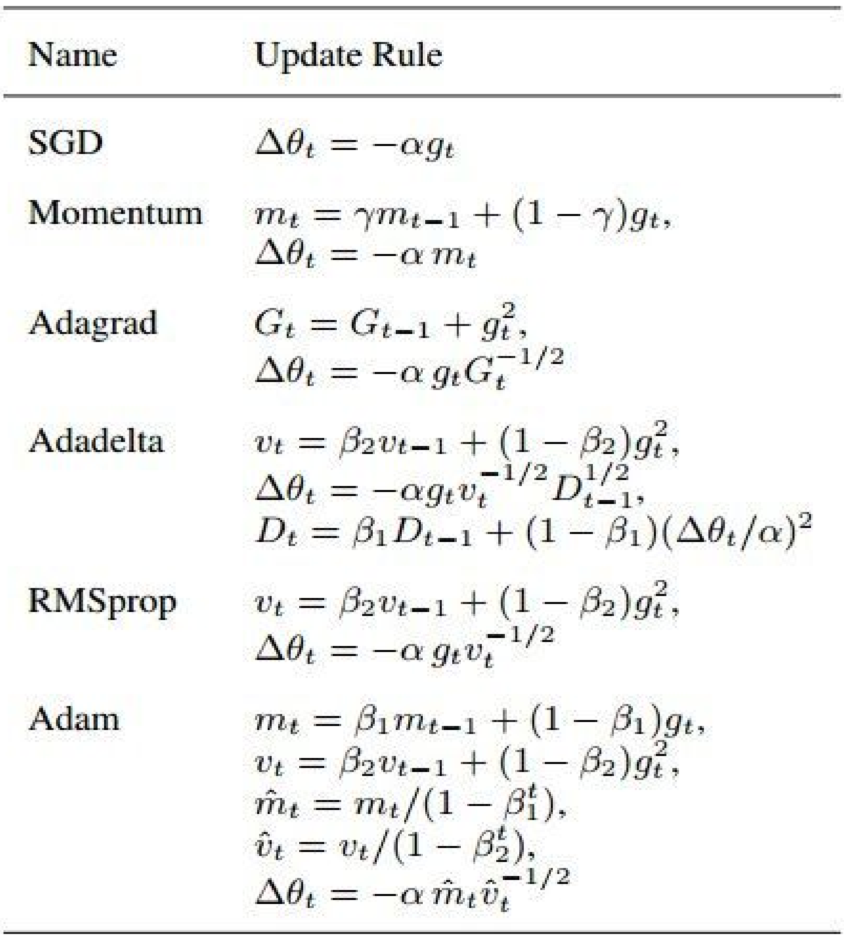
Optimizer : In this model, I used different kinds of optimizers namely : Momentum, Adadelta, ADAgrad, RMSprop, Adam and SGD.

The ADAgrad optimizer essentially uses a different learning rate for every parameter and every time step. The reasoning behind ADAgrad is that the parameters that are infrequent must have larger learning rates while parameters that are frequent must have smaller learning rates.

RMSprop considers fixing the diminishing learning rate by only using a certain number of previous gradients.

Adaptive Moment Estimation, or Adam, computes the adaptive learning rates for each parameter by considering the exponentially decaying average of past squared gradients and the exponentially decaying average of past gradients.

Gradient Descent calculates gradient for the whole dataset and updates values in direction opposite to the gradients until we find a local minima. Stochastic Gradient Descent performs a parameter update for each training example unlike normal Gradient Descent which performs only one update. Thus it is much faster.



Name	Update Rule
SGD	$\Delta\theta_t = -\alpha g_t$
Momentum	$m_t = \gamma m_{t-1} + (1-\gamma) g_t,$ $\Delta\theta_t = -\alpha\, m_t$
Adagrad	$G_t = G_{t-1} + g_t^2,$ $\Delta\theta_t = -\alpha\, g_t G_t^{-1/2}$
Adadelta	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\Delta\theta_t = -\alpha g_t v_t^{-1/2} D_{t-1}^{1/2},$ $D_t = \beta_1 D_{t-1} + (1-\beta_1)(\Delta\theta_t/\alpha)^2$
RMSprop	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\Delta\theta_t = -\alpha\, g_t v_t^{-1/2}$
Adam	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$ $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\hat{m}_t = m_t/(1-\beta_1^t),$ $\hat{v}_t = v_t/(1-\beta_2^t),$ $\Delta\theta_t = -\alpha\, \hat{m}_t \hat{v}_t^{-1/2}$

(Image courtesy : google)

MODEL 2: LONG-SHORT TERM MEMORY

In this model, I have tried an architecture with 100 lstm cells in each LSTM layer, and our LSTM model is composed of a sequential input layer followed by 3 LSTM layers and dense layer with activation and then finally a dense output layer with linear activation function. We have a batch of 100 samples, each sample is a sequence of TIME STEP = 5. Hence we have 1520 sequences to train, with 75% as training data and 25% as test data. I took ideas from this source.

Data Preprocessing: The pre-processing stage involves :

a) Data discretization: Part of data reduction but with particular importance, especially for numerical data

b) Data transformation: Normalization.

c) Data cleaning: Fill in missing values.

d) Data integration: Integration of data files. After the dataset is transformed into a clean dataset, the dataset is divided into training and testing sets so as to evaluate.

Optimizer : The type of optimizer used can greatly affect how fast the algorithm converges to the minimum value. Also, it is important that there is some notion of randomness to avoid getting stuck in a local minimum and not reach the global minimum.

I have chosen to use Adam optimizer, which combines the perks of two other optimizers: ADAgrad and RMSprop. Adaptive Moment Estimation, or Adam, computes the adaptive learning rates for each parameter by considering the exponentially decaying average of past squared gradients and the exponentially decaying average of past gradients.

Regularization: To make sure the weights do not get too large and start focusing on one data point, hence overfitting. I have used Tikhonov regularization.

Dropout : This forces the model to not be over dependent on any groups of neurons, and consider all of them. Dropouts have found their use in making the neurons more robust and hence allowing them to predict the trend without focusing on any one neuron.

PERFORMANCE MEASUREMENT

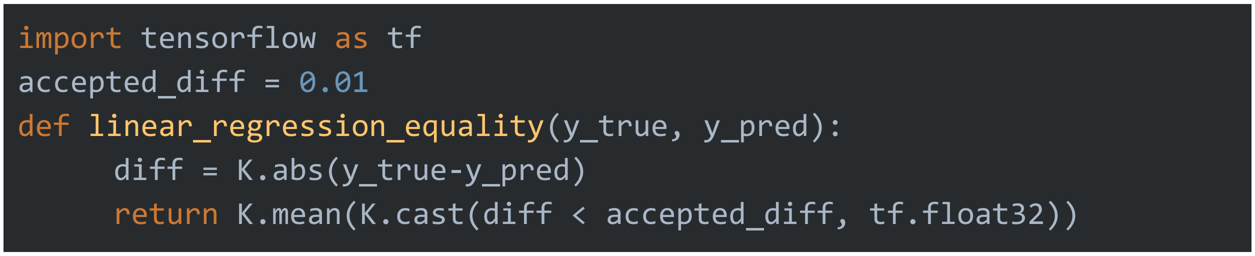
The pricing performance of different models are quantified based on some error measurement parameters. In most of the studies mean squared error is used as a performance measurement criteria and is more intuitive. MSE is the workhorse of basic loss functions: it’s easy to understand and implement and generally works pretty well. To calculate MSE, we take the difference between our predictions and the ground truth, square it, and average it out across the whole dataset. In the next section, I have shown the comparison of different results. But, the results are shown only for one 3, 4 & 5 layer networks.

Comparison of the different results (only for ANN models):

In MODEL 1, all layers are given same activation functions. The table representation shall help us see which combination gives the maximum accuracy for a particular model. Each cell in the table gives the accuracy (in percentage) obtained using that (Activation function, Optimizer) pair. In all cases, the output layer has linear activation function.

One important thing is that the accuracy values can change in different trials, the table gives only one trial instance (and I assume that the accuracy may not change from given values drastically).

In the model, I used 80-20 split for training-validation data. The metric used in training is custom made as shown below:



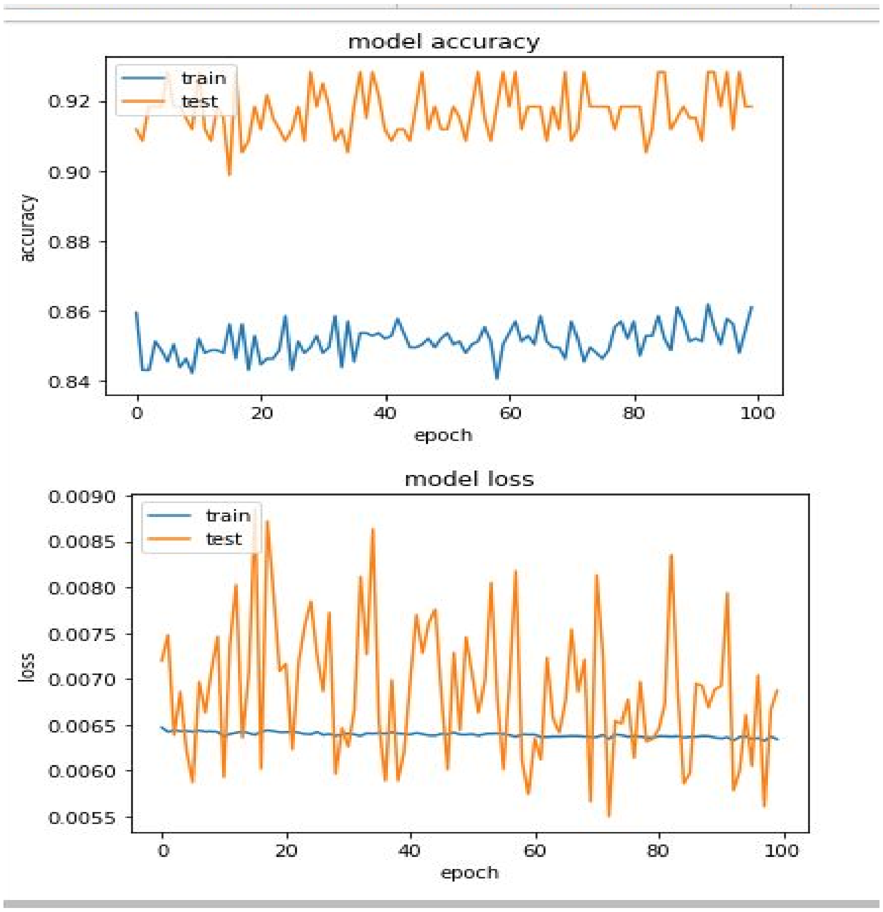
```
import tensorflow as tf
accepted_diff = 0.01
def linear_regression_equality(y_true, y_pred):
      diff = K.abs(y_true-y_pred)
      return K.mean(K.cast(diff < accepted_diff, tf.float32))
```

MODEL 1.A :

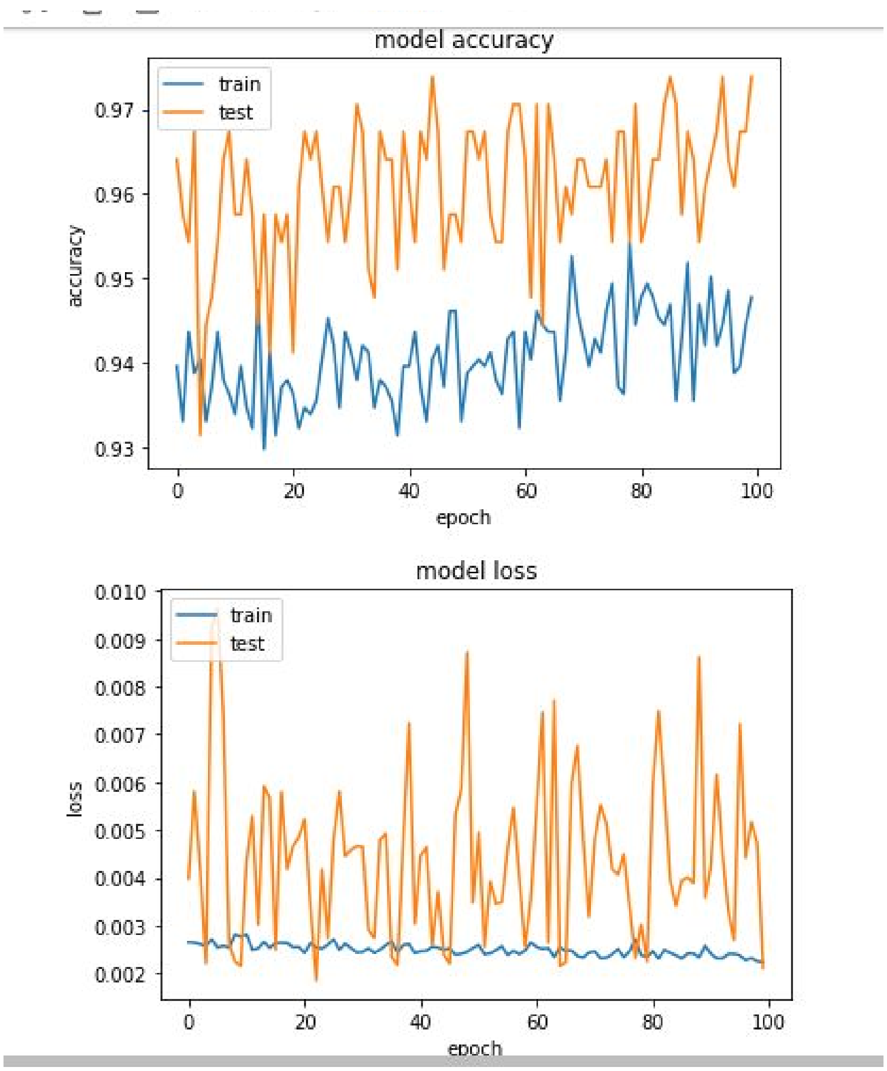
	Adam	Adadelta	Adagrad	Adamax	RMSprop	SGD	Accuracy(max)
sigmoid	79.98	81.94	81.70	81.78	84.07	86.11	86.11
tanh	94.77	92.73	94.20	93.79	89.22	81.13	94.77
RELU	94.28	96.90	97.55	97.30	95.83	97.39	97.55

The train-validation accuracy-loss plots for the maximum accuracy cases only:

Act fun: SIGMOID, Opt : SGD Accuracy Achieved : 86.11

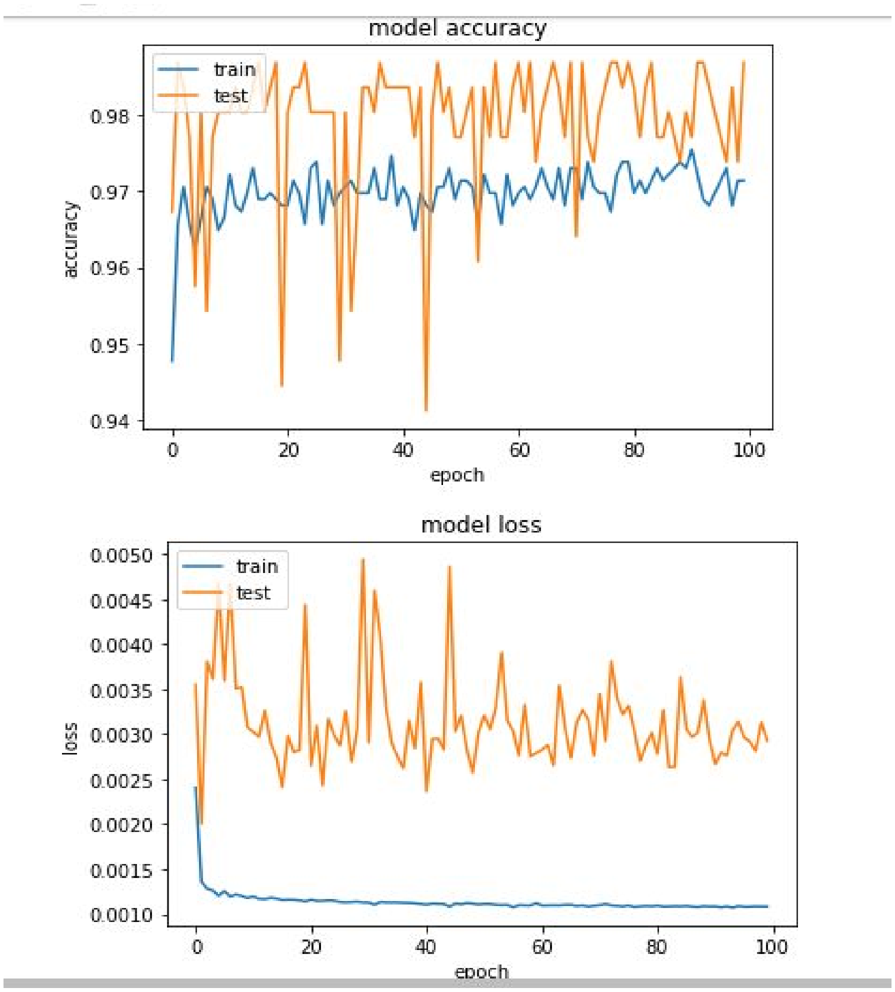


Act fun : TANH Opt : ADAM   Accuracy Achieved : 94.77



Act fun : RELU, Opt : Adagrad

Accuracy Achieved : 97.55

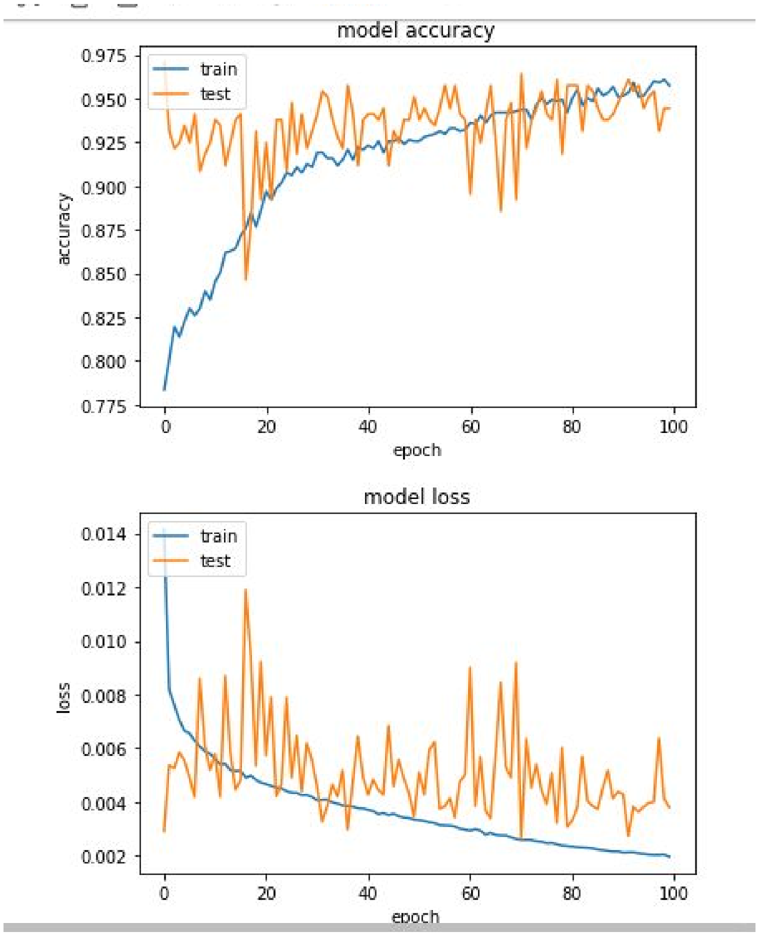


MODEL 1.C :

	Adam	Adagrad	Adadelta	Adamax	RMSprop	SGD	Accuracy (max)
sigmoid	81.62	84.48	85.38	89.30	78.84	75.90	89.30
tanh	91.83	95.51	92.40	93.87	92.08	96.57	96.57
RELU	96.08	98.6	95.67	96.16	95.34	87.01	98.6

Act Fun: RELU, Opt : Adagrad

Accuracy Achieved : 98.6

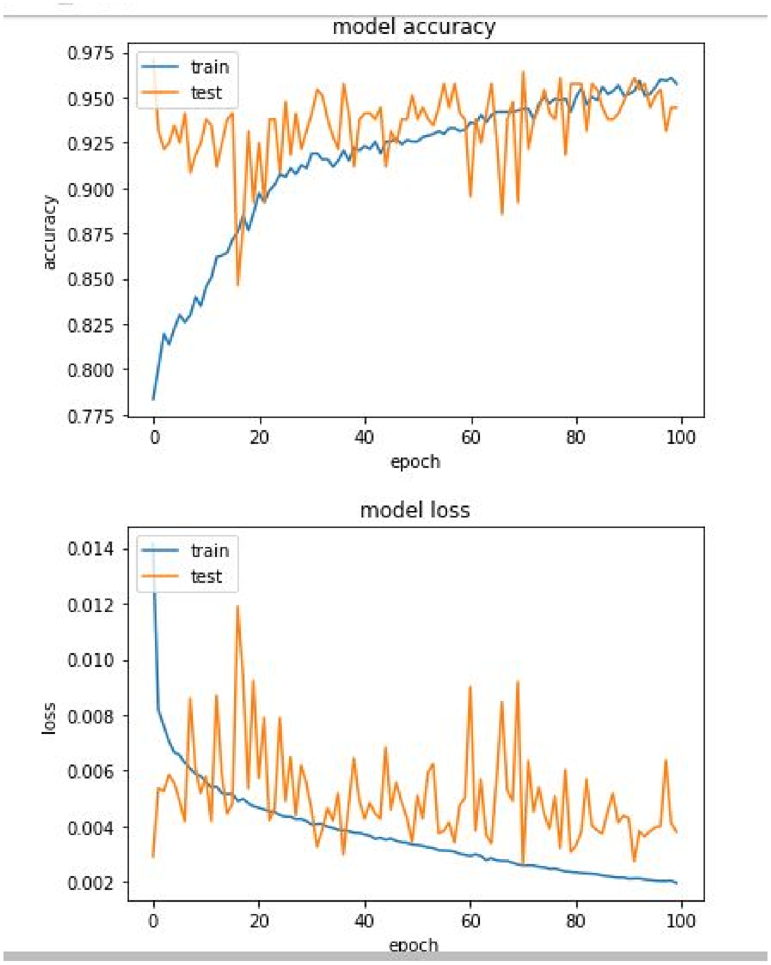


MODEL 1.D :

	Adam	Adagrad	Adadelta	Adamax	RMSprop	SGD	Accuracy (max)
sigmoid	80.72	75.98	78.27	79.25	77.94	56.29	80.72
tanh	91.83	90.60	93.55	94.69	92.73	81.21	94.69
RELU	95.92	97.63	96.49	96.00	96.16	84.80	97.63

Act Fun: RELU, Opt : Adagrad

Accuracy achieved : 97.63



OBSERVATIONS

Firstly, in the above results, we can observe that RELU activation out performs in all three cases. This might be because, RELU is finally linear for positive values. Hence we can conclude that this problem is linear, i.e. it has a linear decision boundary (the regression boundary)

Secondly, ADAM and ADAgrad optimizers give the best results compared to all other optimizers used. Hence, an adaptive optimizer can be preferred over constant rate optimizers like SGD.

Thirdly, we can see that, the maximum accuracy reached is 98.6%. This shows that artificial neural networks are indeed able to approximate a function to fit the given data. Also, if the number of neurons in each layer is increased, or even by increasing the number of layers, the model performs better.

CONCLUSION & FUTURE WORK

It has been more than 40 years since the famous option pricing formula was developed by Black and Scholes. Many improvements to their formula have been done by both the academic researchers and the practitioners since then. One of the critical point for the option pricing theory came with the release of the article with a new approach taken by Hutchinson et al. (1994) who used a neural network method to predict the option price.

In this project, I have tried to analyse the applications of Deep Learning techniques namely **Artificial Neural Networks and LSTM in options pricing**. It is intuitive that, in most cases the neural networks will have better prediction mechanism than the closed-form Black-Scholes formula, since it is very important to incorporate the data into the model, so that the temporal nature of the data is captured, in order to find out the true price of options.

Further research is suggested for experimenting with different input variables that could for example improve the approximation of the network as well as experimenting with different types of options data. Additionally, research in the future might include different types of neural networks such as Convolutional Neural Networks (CNNs) and Spiking Neural Networks (SNNs) into the comparison. The neural network models should also be compared to other types of parametric formulae, such as the ones by Merton (1976) and Heston (1993). Similarly, neural networks could be researched in areas where there are no analytical solutions to compute the price of an option.

BIBLIOGRAPHY

https://www.researchgate.net/publication/220204936_A_neural_network_model_for_estimating_option_prices

https://core.ac.uk/download/pdf/39663999.pdf

https://blog.usejournal.com/stock-market-prediction-by-recurrent-neural-network-on-lstm-model-56de700bff68

https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8

https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c

https://www.researchgate.net/publication/225165990_Comparison_of_new_activation_functions_in_neural_network_for_forecasting_financial_time_series

http://jultika.oulu.fi/files/nbnfioulu-201901091016.pdf

https://stackoverflow.com/questions/42665359/how-do-you-compute-accuracy-in-a-regression-model-after-rounding-predictions-to (for custom made metric in keras)

https://stackoverflow.com/questions/50797135/keras-network-acc-zero-and-loss-very-low

- (the reason for need for custom made metric for learning)

https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/

http://www.scientific-consultants.com/nnbd.html (DATA) https://www.youtube.com/watch?v=ca7oC70BnTg

https://www.khanacademy.org/economics-finance-domain/core-finance/derivative-securities/black-scholes/v/introduction-to-the-black-scholes-formula

https://www.cse.iitb.ac.in/~saketh/research/DeodaDDP.pdf

https://www.asx.com.au/prices/pricing_models.htm

http://www.cboe.com/products/vix-index-volatility/volatility-on-stock-indexes/cboe-nasdaq-100-volatility-index-vxn

https://www.investopedia.com/articles/optioninvestor/06/newvix.asp

https://www.edupristine.com/blog/financial-instruments

https://www.investopedia.com/articles/optioninvestor/05/020205.asp

https://www.investopedia.com/articles/optioninvestor/10/etf-options-v-index-options.asp

https://www.thebalance.com/trading-stock-indexes-1031061

https://www.cse.iitb.ac.in/~saketh/research/DeodaDDP.pdf

https://pure.bond.edu.au/ws/portalfiles/portal/18243185/Option_Pricing_Using_Artificial_Neural_Networks.pdf

https://helda.helsinki.fi/dhanken/bitstream/handle/123456789/202968/huynh.pdf?sequence=1&isAllowed=y

https://www.weareworldquant.com/en/thought-leadership/beyond-black-scholes-a-new-option-for-options-pricing/

https://medium.com/@ranko.mosic/option-pricing-and-volatility-forecasting-using-deep-learning-4a57ebb1f39