Deep Learning for Option Pricing

MA473 Project

Dev Priya Goel, Shruti Agarwal and Kartik Sethi

1 Introduction

In computational finance, numerical methods are commonly used for the valuation of financial derivatives and also in modern risk management. Different numerical methods such as Monte Carlo simulations, finite differences, Fourier methods are often used to solve option price PDEs. Instead of predicting option prices as is the norm, we choose to focus on implied volatility which is another important parameter for predicting asset behaviour and is intrinsically useful to option price predictions. Among all the variables in the Black Scholes model, volatility is the most crucial one. Unlike other variables, volatility cannot be observed directly from the market and thus must be estimated. It is well known that estimating volatility of the underlying asset is the most difficult aspect of option valuation.

The reason for choosing volatility is that predicting options directly generally yields poor results due its high dependence on underlying asset price S and strike price K. Option prices may vary from Rs. 5 to Rs. 5000 in a day depending on K. (This is the case for out dataset). Therefore, the error in this case is high.

Models which predict volatility using usually use time series. They sample out one option per day and use Black Scholes to predict the option price or some another estimator. The problem with this approach is that for dataset like ours, there our multiple contracts in a day and using the same estimate of volatility for each of them to compute option prices doesn't yield accurate results. The well known metric - implied volatility, for option contracts, varies a lot even within the same day for different contracts which we discovered while experimenting with the dataset. Thus, time-series prediction won't be applicable if we are working with implied volatility.

The groundbreaking work of Black and Scholes (1973) on option pricing assumes that all option prices on the same underlying asset with the same expiration date and different exercise prices should have the same implied volatility (Canina and Figlewski, 1993). However, studies have found that implied volatility tends to differ across moneyness and time to expiration. This is well inferred in the US market on stock indices after the October 1987 crash (Rubinstein, 1994; Jackwerth and Rubinstein, 1996)[1]. Therefore, we used a feature estimator approach.

At this point one may question, how to capture the time dependence of the volatility without time series? For that explicit purpose, some cleverly chosen features with LSTM model came to our rescue. We use historical 20-day underlying asset price data and log returns. Also, we used LSTM to extract temporal dependence features. Many GARCH models don't go beyond 20-day so LSTM, a state-of-the-art model should be able to capture the time series component through this data. Our method combines best of both worlds: (1) feature learning based regression and (2) time series incorporation through LSTM with historical 20-day log returns and stock prices.

Our report is organised as follows: In **section 1** we briefly introduce our methodology and give a theoretical background on Black Scholes PDE and Implied Volatility; in **section 2** we talk about deep learning in general and the neural network architectures we have used in our prediction model; in **section 3** we touch upon the dataset that we have used i.e. NIFTY50 European option contracts (put and call); in **section 4** we go into details of our methodology; in **section 5** we introduce the performance measures that we have used to evaluate our model; in **section 6** present all our experiments and their results; in **section 7** we briefly discuss our observations of the year 2020 and the impact of COVID-19 on Indian market; in **section 9** we conclude our report and talk about its future scope; in **section 10** we provide a guide to our code.

1.1 Black Scholes PDE

One of the most prominent methods for modelling asset prices is the Geometric Brownian Motion (GBM) model. The GBM model gives rise to the Black-Scholes PDE for calculation option prices. Asset prices under GBM are given by

$$dS_t = \mu S_t dt + \sqrt{\nu} S_t dW_t \tag{1}$$

where S is the price of a non-dividend paying asset, and W is a Wiener process, with t being the time, μ the drift parameter, and ν the variance parameter. The volatility parameter is $\sigma = \sqrt{\nu}$. A European option contract on the underlying stock price can be valued via the Black-Scholes PDE, which can be derived from Itô's Lemma under a replicating portfolio approach or via the martingale approach. Denoting the option price by $V(t, S)$, the Black-Scholes equation reads,

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial t^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{2}$$

with time t until to maturity T, and r the risk-free interest rate. We also have the terminating boundary condition for specific options. For example, the payoff for European call is given by

$$V(t = T, S) = (S - K)^+ \tag{3}$$

where K is the option's strike price. An analytic solution to 2, 3 exists for European plain vanilla options, i.e.,

$$V_c(t, S) = SN(d_1) - Ke^{-r\tau}N(d_2) \tag{4}$$

$$d_1 = \frac{log(S/K) + (r - 1/2\sigma^2)\tau}{\sigma\sqrt{\tau}}, \quad d_2 = d_1 - \sigma\sqrt{\tau}, \tag{5}$$

where $\tau := T - t, V_c(t, S)$ is the European call option value at time t for stock value S, and $N()$ represents the normal distribution.

Similarly, the payoff for European put is given by

$$V(t = T, S) = (K - S)^+ \tag{6}$$

where K is the option's strike price. An analytic solution to 2, 6 exists for European plain vanilla options, i.e.,

$$V_p(t, S) = Ke^{-r\tau}N(-d_2) - SN(-d_1) \tag{7}$$

$$d_1 = \frac{log(S/K) + (r - 1/2\sigma^2)\tau}{\sigma\sqrt{\tau}}, \quad d_2 = d_1 - \sigma\sqrt{\tau}, \tag{8}$$

where $\tau := T - t, V_p(t, S)$ is the European put option value at time t for stock value S, and $N()$ represents the normal distribution.

1.2 Implied Volatility

Given an actual market option price V^{mkt}. the Black-Scholes implied volatility σ^* can be determined by solving $BS(\sigma^*,\ S, K, \tau, r) = V^{mkt}$. The monotonicity of the Black-Scholes equation with respect to the volatility guarantees the existence of $\sigma^* \in [0, \infty]$. We can write the implied volatility as an implicit formula by inverting the Black Scholes equation.

$$\sigma^*(K, T) = BS^{-1}(V^{mkt}; S, K, \tau, r) \tag{9}$$

One of the limitations of the Black Scholes model is the assumption of constant volatility σ. When analysing practical financial data, it was seen that volatility actually varies with time and can be thought of as a stochastic process. The most important evidence for considering volatility to be stochastic is the occourence of implied volatility smile/skew. It is seen that when we calculated implied volatility for actual asset data, the implied volatility does not remain constant, infact a volatility smile/skew curve is observed.

1.2.1 Numerical Methods for Implied Volatility

The inverse of Black Scholes equation does not have a analytical solution, thus we need to use numerical methods for solving 9. One of the most common iterative methods for finding roots of a equation is Newton-Rhapson method. The Newton-Rhapson iteration for finding the implied volatility is given by

$$\sigma^*_{k+1} = \sigma^*_k - \frac{V(\sigma^*_k) - V^{mkt}}{g'(\sigma^*_k)} \quad , k = 0 \cdots \tag{10}$$

where $g(\sigma^*) = BS(\sigma^*,\ S, K, \tau, r) - V^{mkt}$

2 Deep Learning

Machine and deep learning-based algorithms are the emerging approaches in addressing prediction problems in the financial sector. These techniques generally produce more accurate results than conventional regression-based modeling. Extensive research shows that artificial Recurrent Neural Networks (RNN) with memory, such as Long Short-Term Memory (LSTM), provide superior performance compared to Autoregressive Integrated Moving Average (ARIMA) by a large margin. This can be credited to the additional "gates" LSTM-based models incorporate for the purpose of memorizing longer sequences of input data. The gates incorporated in the LSTM architecture offer a good prediction. Bidirectional LSTMs (BiLSTMs) enable additional training by traversing the input data twice from left-to-right, and right-to-left. BiLSTM, with additional training capability, outperforms regular unidirectional LSTM.

Batch Normalization: Let $\boldsymbol{B}$ be a mini-batch during the training process, the BN layer performs the following operation:

$$\begin{aligned}
\boldsymbol{\mu_B} &= mean(\boldsymbol{B}) \\
\boldsymbol{\sigma_B^2} &= var(\boldsymbol{B}) \\
\boldsymbol{\mu_{mov}} &= \alpha\boldsymbol{\mu_{mov}} + (1-\alpha)\boldsymbol{\mu_B} \\
\boldsymbol{\sigma^2_{mov}} &= \alpha\boldsymbol{\sigma^2_{mov}} + (1-\alpha)\boldsymbol{\sigma^2_B} \\
\boldsymbol{B} &= \frac{\boldsymbol{B} - \boldsymbol{\mu_{mov}}}{\boldsymbol{\sigma_{mov}}} \\
\boldsymbol{B} &= \gamma\boldsymbol{B} + \beta
\end{aligned}$$

where α is a layer parameter called "momentum" whereas, γ and β are trainable parameters.
During test/validation however, the layer skips first 4 steps and uses the variables obtained from training to normalize and scale.

Leaky Relu Activation: It is one of the generalizations of the ReLU activation i.e. $g(z_i) = max\{0, z_i\}$. Unlike ReLU, leaky ReLU uses a non-zero slope α_i when $z_i < 0$, i.e.

$$g(z_i) = max\{0, z_i\} + \alpha_i min\{0, z_i\}$$

We use Leaky Relu activation in our Feed Forward Network explained in below.

2.1 Multilayer Perceptron

Multilayer Perceptron (MLP) or Feed Forward Neural Network is a kind of primary artificial neural network which consists of a finite number of continuous layers. One MLP at least has three layers including input layer, hidden layer, and output layer. In many cases, there could be more hidden layers in the MLP structure which can deal with approximate solutions for many complex problems such as fitting approximation. One MLP can be thought of as a directed graph mapping a set of input vectors to a set of output vectors, which consists of multiple node layers connected to the next layer. These connections are generally called links or synapses. In addition to the input nodes, each node is a neuron with a nonlinear activation function. A supervised learning method (BP algorithm) called backpropagation algorithm is often used to train MLP. MLP is a generalization of the perceptron, which overcomes the weakness that the perceptron can't recognize inseparable linear data. The layers of MLP are fully connected, which means every neuron of each layer is connected with all neurons of the previous layer, and this connection represents a weight addition and sum. 1 shows the framework of MLP model.

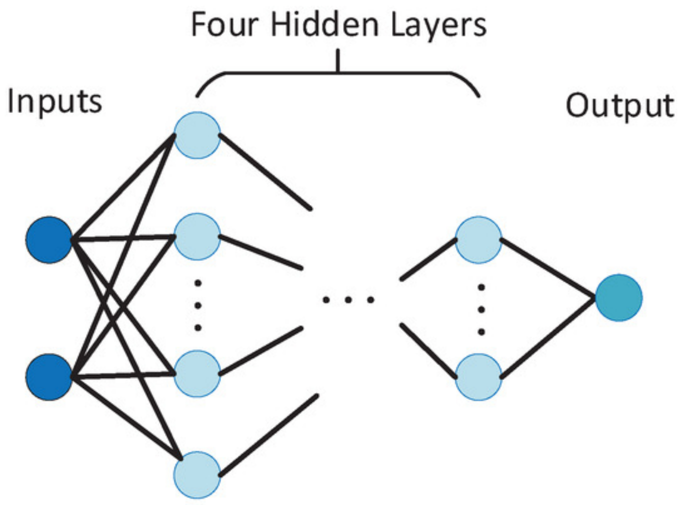


Figure 1: Multilayer Perceptron (MLP) model structure with four hidden layers.

In detail, from the input layer to the hidden layer: X represents the input and H 1 represents the first hidden layer, then the formulation is

$H_1 = f(W_1 X + B_1)$, where W_1 stands for weight parameters and B_1stands for bias parameters, and the function f is usually a non-linear activation function. During the hidden layers, it repeats the similar methods to get the hidden layers' output. Every layer has different weight parameters and bias parameters. From hidden layer to output layer, Y is used to represent the output, then $Y = G(W_L H_L + B_L)$, H_L stands for the last hidden layer and L stands for the number of the hidden layer. The function G is the activation function, commonly known as Softmax. Therefore, the relationships between input and output will be built. If the weight w or bias b of one neuron or several neurons are changed a little bit, the output of that neuron will be different, and those differences will eventually be reflected in the output. Finally, BP algorithm and optimization algorithm are used to update the weight W and bias B to get the expected results

2.2 LSTM

Long Short Term Memory (LSTM) is a particular Recurrent Neural Network (RNN),which successes the superiority of RNN and has better performance with long-term dependency applied in many fields. Due to the vanishing gradient problem, RNN has difficulty in learning long-term dependencies. LSTM is a practical solution for combating vanishing gradients by using memory cells. All recurrent neural networks have a repeated chain in one single neural cell. Traditional RNN only has one layer in the neural cell, but LSTM has four layers which are particularly interacting with each other. The key to LSTM is the cell state C_t. Cell state is like a conveyor belt that runs through the chain, with only linear interaction, keeping information flow unchanged. LSTM can delete or modify the information of the cell state, which is regulated by the gate mechanism. The gate mechanism is a way for information to pass selectively, consisting of the sigmoid neuronal layer and the pointwise multiplication operation. There are three types of gates in the LSTM cell to protect and control cell state. Every gate has an expression of $\sigma(W_i x + b_i)$. The range of sigmoid layer output is (0,1), which indicates how much of each component in C_{t-1} should be passed. If the output is 0, it means that no pass is allowed while the output of 1 representing all pass. Figure 3 shows the framework of LSTM model.

Bidirectional LSTM

LSTM cells predict the current status based only on former information. It is clear that some important information may not be captured properly by the cell if it runs in only one direction. The improvement in bidirectional LSTM is that the current output is not only related to previous information but also related to subsequent information. Bidirectional LSTM is made up of two LSTM cells, and the output is determined by the two together.

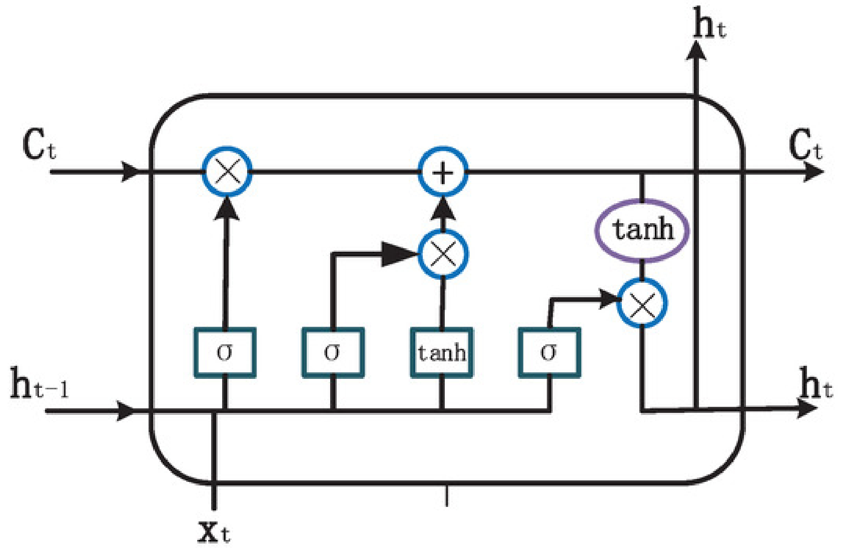


Figure 2: Long Short Term Memory (LSTM) model structure. C_t is the cell state and h_t is the hidden state. σ represents the sigmoid activation function and tanh represents the tanh activation function. $\oplus$ means the element-wise addition and $\otimes$ means element-wise multiplication operation.

2.2.1 Forget Gate Layer

The first step of LSTM is to decide what information to discard from the cell state. This decision is made by a sigmoid layer called the 'forget gate layer'. It will output a vector ranging from 0 to 1 for cell state C_{t-1} according to h_{t-1} and x_t . If the value equals to 1, it means to keep the whole information from C_{t-1} , while 0 indicating dropping the information completely. Here f_t represents the value of the forget gate.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

.

2.2.2 Input Gate Layer

The second step is to decide what new information should be stored in cell state, which consists of two parts:

(1) One sigmoid layer determines what kind of values should be updated represented as i_t .

(2) One tanh layer will create a new candidate value of $\hat{C}_t$ which will be added to the state information.

$$\begin{aligned} i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\ \hat{C}_t &= \tanh(W_c[h_{t-1}, x_t] + b_c) \\ C_t &= f_t * C_{t-1} + i_t * \hat{C}_t \end{aligned}$$

Flowing, the two values of i_t and $\hat{C}_t$ will be combined to update the state. After that, the state C_t by adding two items with $f_t * C_{t-1}$ and $i_t * \hat{C}_t$

2.2.3 Output Gate Layer

Lastly, the output is going to be determined by the output gate layer, which is based on the cell state. The hidden state is the element-wise product of a tanh activation layer of C_t and output gate o_t capturing the information of previous hidden state h_{t-1} and input x_t .

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o), \\ h_t &= o_t * \tanh(C_t) \end{aligned}$$

One fully connected layer will be applied to generate the final predictions based on the hidden states h

3 Dataset

We compiled our dataset by going through NSE's website and downloading historical option prices corresponding to different index prices [2].We have used the NIFTY50 index option for our analysis. The index is the largest and most liquid among Indian securities. It contains 50 of the approximately 1600 companies traded on NSE, the index captures approximately 65% of its float-adjusted market capitalization and is an accurate reflection of the Indian stock market. The NIFTY 50 is owned and managed by NSE Indices Limited [3].

The NIFTY 50 includes the key sectors of the Indian economy and offers investment managers exposure to the Indian market in one efficient portfolio. The NIFTY 50 has been trading since April 1996 and is appropriate for benchmarking index funds and index derivatives. Our data sample includes data from Jan 2015 - Dec 2020.

To find the appropriate risk-free rate r, we matched the yield on the 3-month bond yields traded in the Indian Market, having maturity closest to the time until expiration of each option [4]. This is widely accepted practice in options trading.

The reason for using historical data is that in many cases the options data may not follow the Black Scholes equation and we want our Neural Network to be able to generalize to these rare events such as the volatility smile.

4 Methodology

Our primary goal was to predict option price using available financial data like historical volatility, price of underlying asset, strike price, treasury rate, time to maturity, etc.. However, when we tried to train the model using above features and predict the option price, the loss was quite high. This can be accredited to the fact the our data had a huge variety of options as a result of varied strike prices and the model was not able to learn the sensitivity of option price to strike price.

Our approach to resolving this was simple yet fruitful. We used the model to predict implied volatility using (9) and then substituted this into the Black-Scholes equation to calculate the option price.

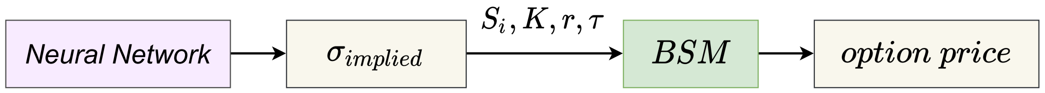


Figure 3: Methodology

After an extensive literature review ([5], [6], [7]) and a lot of trial-and-error we arrived at the following set of features for training our model -

Feature (Ephemeral)	**Description**
S_i	The price of the underlying asset
K	The strike price
r	The returns on a government bond
σ_H	Historical volatility using past 20-day data
$R_i = \log(S_i/S_{i-1})$	Log return
σ_{CAL}	Calibrated volatility using past 20-day data
τ	Time to maturity (in years)
$\text{moneyness}_1 = S_i/K$	Ratio of underlying asset price to strike price
$\log(K/S)$	
$\log(K/S)\tau^{-0.5}$	
$\log(K/S)\tau^{-0.95}$	
Feature (Temporal)	**Description**
$S_{i-19}, \ldots, S_{i-1}, S_i$	Past 20-day price of underlying asset

Table 1: Input Data

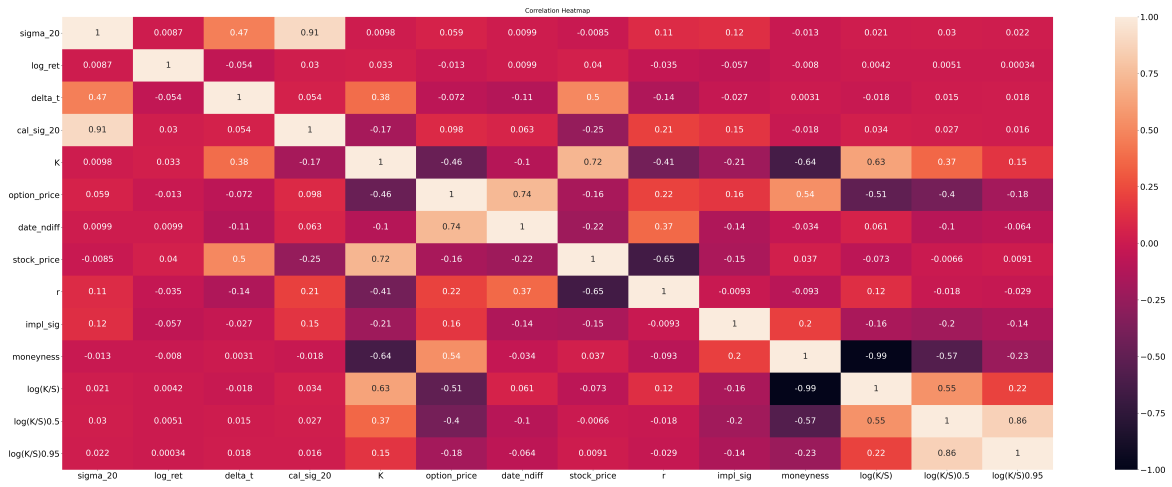


Figure 4: Correlation Heatmap for the Dataset

Also, the neural network architecture for predicting implied volatility -

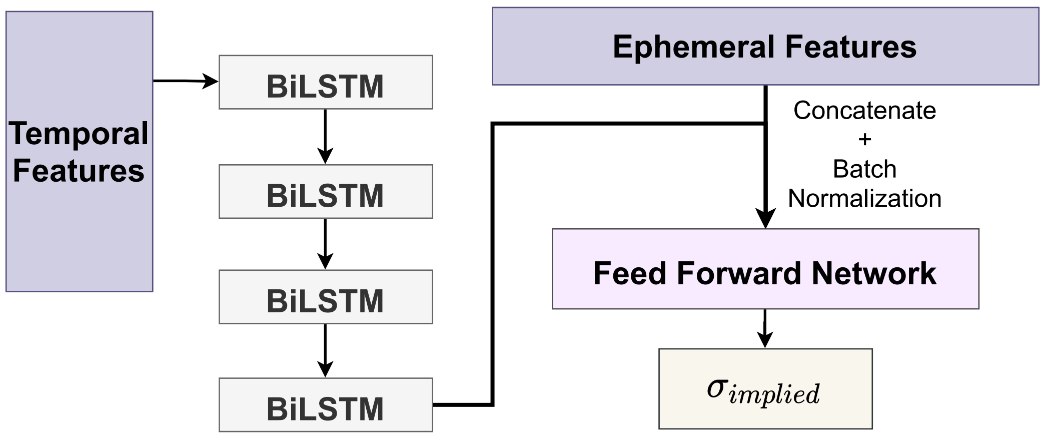


Figure 5: Neural Network Model for predicting Implied Volatility using features given in Table 5.

5 Performance Measures

In this part, the accuracy measurements would be introduced which are designed to compare the performance of the four different models. Here, we choose three classical indicators including Mean Absolute Percentage Error (MAPE), Root Mean Square Error (RMSE), and Correlation Coefficient (R) as the measurements for the predictive performance of each model. The equation of each indicator is shown in the following section:

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

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

$$R = \frac{\sum_{i=1}^{N}(y_i - \bar{y}_i)(\hat{y}_i - \bar{\hat{y}}_i)}{\sqrt{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2(\hat{y}_i - \bar{\hat{y}}_i)^2}}$$

In the above equations, y_i is the original series that we want to forecast and $\bar{y}_i$ is the mean of the original series. $\hat{y}_i$ is the predicted series obtained from the model output and $\bar{\hat{y}}_i$ is the mean of the predicted series. N represents the number of samples in one batch. Among the three indicators, MAPE measures the size of the error calculated as the relative average of the error, and RMSE measures the mean square error of the true values and the predicted values. R is a measure of the linear correlation between two variables. Here, the model would perform well with smaller MAPE and RMSE. The predicted series is similar to the actual series with larger R.

6 Experiments

6.1 Experiment 1: Setup

In this experiment, we started out by training the model on the complete NSE dataset for European call option.

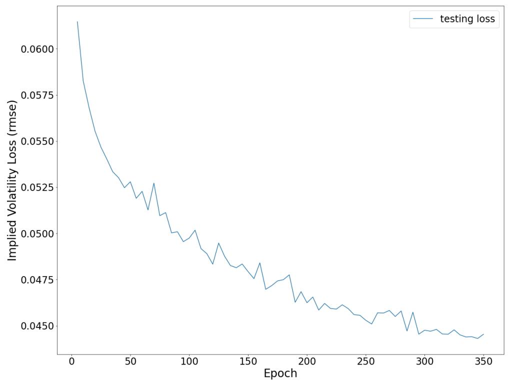
6.2 Experiment 1: Results

Results of Experiment 1 are as follows -

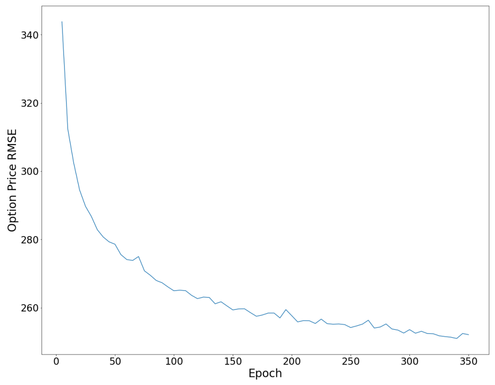
Implied Volatility	Option Price		
RMSE	RMSE	MAPE	R
0.0443	252.462	185.205	0.978

Table 2: Results of Experiment 1

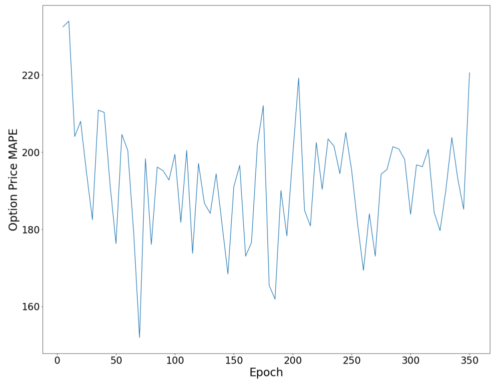
Following graphs will give more insight into the learning curve of the model over the epochs



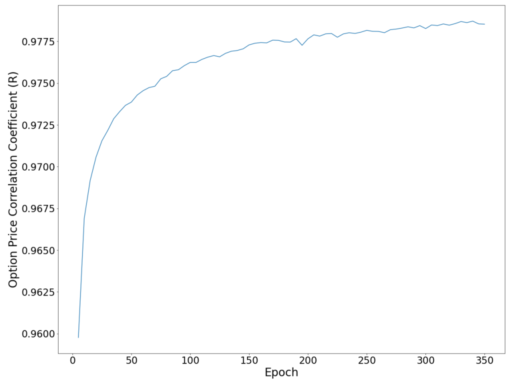
(a) Implied Volatility RMSE



(b) Option Price RMSE



(c) Option Price MAPE



(d) Option Price R

At this point, MAPE for option price was quite high. Even after extensive training, it wasn't showing any trends (decline). This can be seen from the figures below.

6.3 Experiment 2: Setup

To counter the shortfalls of Experiment 1, we segregated the dataset based on moneyness$_1$ (defined by S_i/K) into- (1) at-the-money, (2) in-the-money and (3) out-the-money and trained for all the three separately.
Let us define at-the-money, in-the-money and out-the-money options -

Option Type	**Definition**
In-the-money	$S/K > 1.05$
At-the-money	$0.97 \leq S/K \leq 1.05$
Out-the-money	$S/K < 0.97$

Table 3: Option Type and their definitions

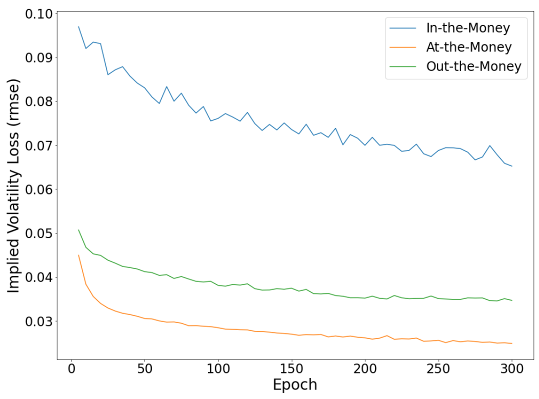
6.4 Experiment 2: Results

The results are given as follows -

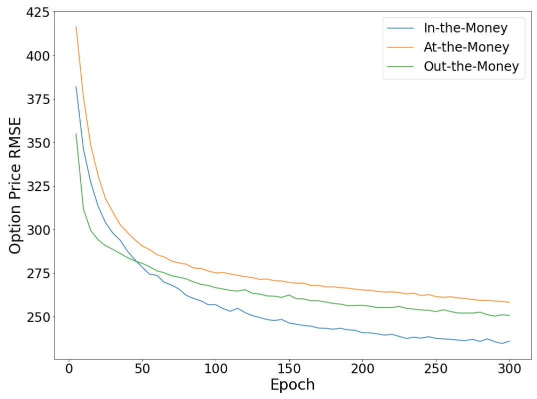
	Implied Volatility	**Option Price**		
	RMSE	**RMSE**	**MAPE**	**R**
In-the-Money	0.0658	234.629	8.537	0.983
At-the-Money	0.0249	258.997	27.994	0.969
Out-the-Money	0.0347	250.820	372.67	0.961

Table 4: Results for Experiment 2

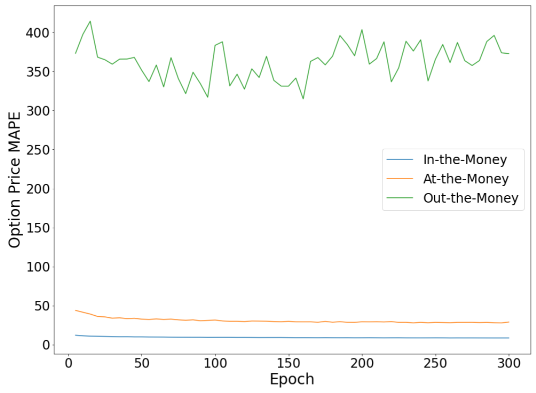
Following graphs will give more insight into the learning curve of the model over the epochs -



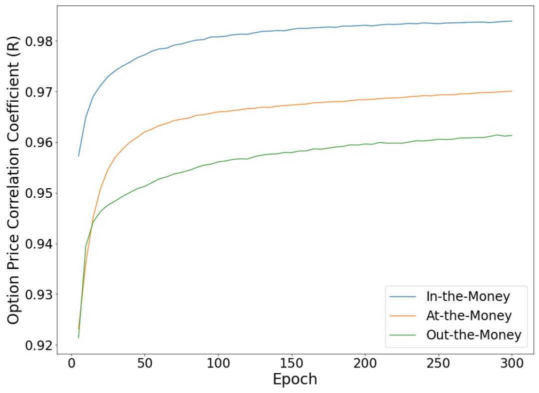
(e) Implied Volatility RMSE



(f) Option Price RMSE



(g) Option Price MAPE



(h) Option Price R

The model showed significant improvement for at-the-money and in-the-money options. However, out-the-money did not showed much improvement.

6.5 Experiment 3: Setup

To deal with the problems in the previous experiments, we set up a new model with added features:

Feature (Ephemeral)	**Description**
Sharpe ratio	$\mathbf{E}[R_a - R_b]/\sigma_a$
	R_a = log return of the asset
	R_b = Risk-free return
	σ_a = standard deviation of the asset excess return
moneyness$_2 = S_i e^{r\tau}/K$	Ratio of future price to strike price
arbitrage indicator	option price $-$ max((future price $- K)e^{-r\tau}, 0)$
Feature (Temporal)	**Description**
$R_{i-19}, \ldots, R_{i-1}, R_i$	Past 20-day returns on asset

Table 5: Input Data

Alongside this, we also exclude outliers hindering the learning of our models.

Eviction of Outliers

Following exclusion criteria has been used on the option pricing data for increased accuracy [8] -

1. Expiry: Contracts having maturity less than 6 days [trading days] to maturity are excluded. Short term option contracts are very sensitive to the non-synchronous price issues and driven by market sentiments.

2. moneyness$_2$ (m): Option moneyness$_2$ (m) is defined as ratio of future price of underlying security to option strike price.

$$m = \frac{F(t, \tau)}{K}$$

   where, $F(t, \tau)$ is future price of NIFTY at time t of expiry τ years and K is the option strike price. Option contracts trading at highly deep in-the-money contains very less information about volatility. Therefore, in the present study highly deep in-the-money option contracts ($m > 1.10$) has not been considered for the empirical investigation. Similarly, option contracts trading at deep-out-money, for $m < 0.9$, has not been considered for empirical investigation.

3. Arbitrage Opportunities: European call option is said to provide arbitrage opportunity, if

$$c(t, \tau) > \max\{e^{-r\tau}(F(t, \tau) - K), 0\}$$

   where, $c(t, \tau)$ is the option price at time $t, F(t, \tau)$ is future price of NIFTY having same maturity as option, and τ is expiry period. So, option contracts not satisfying arbitrage condition have excluded from the study.

In this setup, the data was segregated based on moneyness$_2$ (m) into -

Option Type	**Definition**
Deep out-the-money	$m < 0.95$
Out-the-money	$0.95 \leq m < 0.98$
Near out-the-money	$0.98 \leq m < 1.00$
Near in-the-money	$1.00 \leq m < 1.02$
In-the-money	$1.02 \leq m < 1.05$
Deep in-the-money	$1.05 \leq m \leq 1.10$

Table 6: Option Type and their definitions

6.6 Experiment 3: Results

Experiment 3 performed orders of magnitude better than the other two experiment. The losses in all the scenarios were quite low. As this was our best trial yet we performed it for both call and put options -

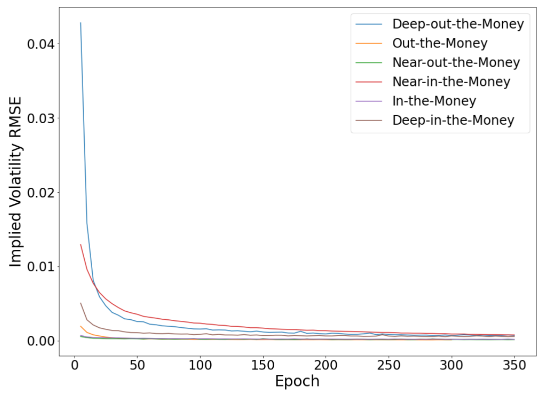
6.6.1 For European Call Option

The results from the experiment can be seen in the graphs below.

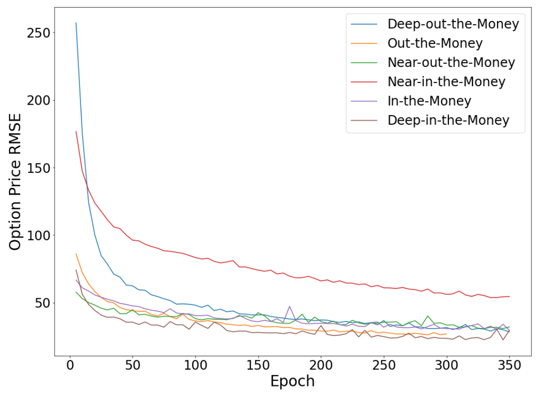
	Implied Volatility	**Option Price**		
	RMSE	**RMSE**	**MAPE**	**R**
Deep-Out-the-Money	0.000701	29.071	46.216	0.999
Out-the-Money	0.000110	26.390	8.922	0.999
Near-Out-the-Money	0.000127	30.928	5.592	0.999
Near-In-the-Money	0.000798	53.943	5.131	0.998
In-the-Money	0.000171	29.652	1.982	0.999
Deep-In-the-Money	0.000573	29.627	30.785	0.999

Table 7: Results of Experiment 3: European Call Option

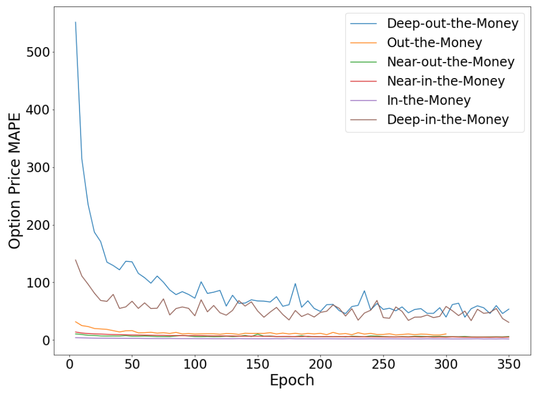
Following graphs will give more insight into the learning curve of the model over the epochs -



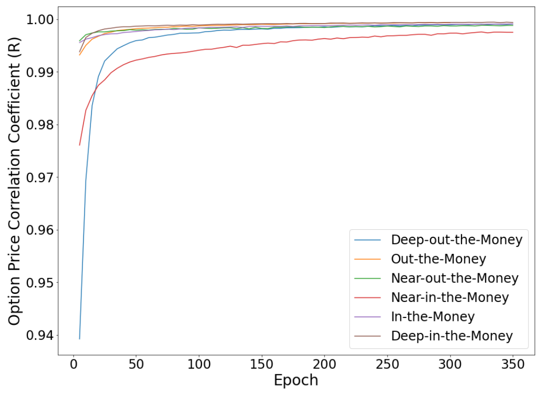
(i) Implied Volatility RMSE



(j) Option Price RMSE



(k) Option Price MAPE



(l) Option Price R

6.6.2 For European Put Option

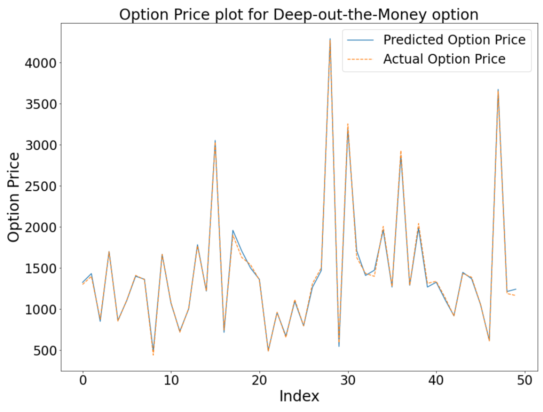
The results from the experiment are as follows:

	Implied Volatility	**Option Price**		
	RMSE	**RMSE**	**MAPE**	**R**
Deep-Out-the-Money	0.000720	34.976	1.848	0.998
Out-the-Money	0.000178	373.84	52.38	0.959
Near-Out-the-Money	0.000386	35.440	4.025	0.996
Near-In-the-Money	0.001486	63.166	7.770	0.989
In-the-Money	0.000101	38.358	4.344	0.994
Deep-In-the-Money	0.000505	28.109	1.431	0.999

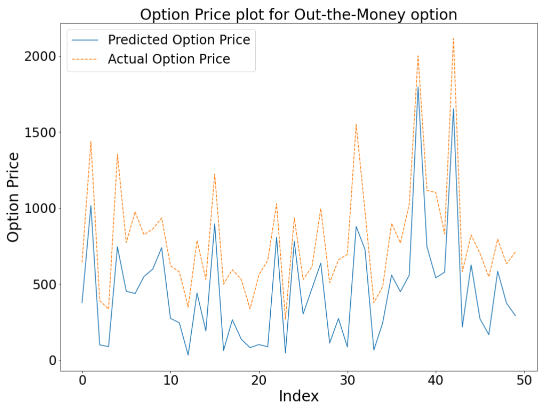
Table 8: Results of Experiment 3: European Put Option

6.7 Experiment 3: Predictions

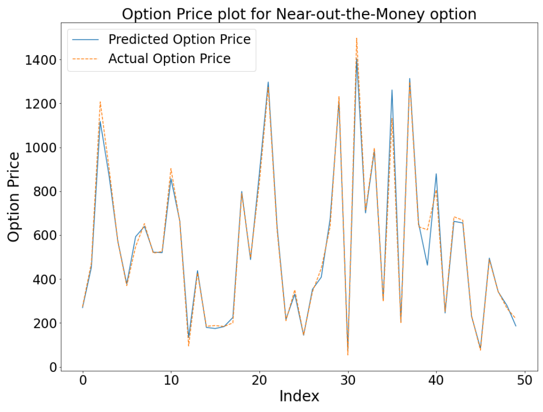
Following is the comparison of actual option price to the predicted option price based on the segregation for a few of the options from the test set for european put option.



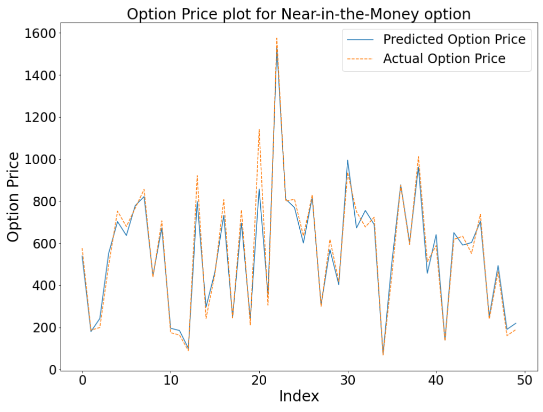
(m) Deep out-the-money



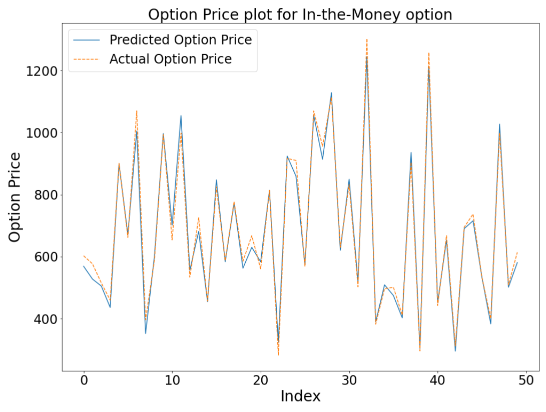
(n) Out-the-money



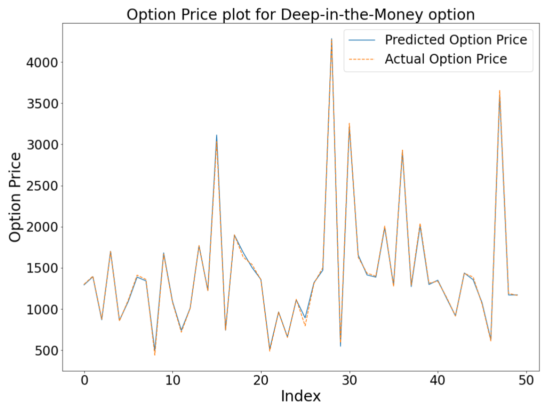
(o) Near out-the-money



(p) Near in-the-money



(q) In-the-money



(r) Deep in-the-money

7 Special Analysis: Impact due to COVID-19

2020 was a bizarre year for the Indian market with COVID followed by a ton of investments from overseas. Most of the out-of-money options were traded in our NSE dataset were from this period. From the experiments we have seen that prices for these options are notoriously had to predict.

8 Conclusion

Limited research on option pricing is available for India owing to the nascent nature of its derivatives market. Existing work provides evidence for the existence of a volatility smile in the Indian options market.
Owing to this, we took it upon ourselves to predict the implied volatility and correspondingly to predict the option price. We used a method that combines best of both worlds: (1) feature learning using regression and (2) time series element incorporation by using LSTM on historical 20-day log returns and stock prices. The model was quite successful in predicting option prices for Eurpean put and European call options.
In future, we hope to test the robustness of model in pricing real data for American and Asian options.

9 Our Code

Since the third experiment had the most features and outperformed the other two by a huge margin, we have included only that code in our final submission. The datasets after pre-processing and adding all the feactures were quite huge (call 462 MB and put 262 MB). So, we have added it to our drive folders for submission.
For running the modelling code:

1. Download both the pre-processed datasets `new_model_final_put_df.csv` and `new_model_final_call_df.csv` from here or here. Keep the downloaded files in the same folder as the code.

2. Run the notebook `Call.ipynb` for modelling and analysing European call option price prediction and run the notebook `Put.ipynb` for modelling and analysing European put option price prediction.

We have also added the data processing code for reference. For this code:

1. Download all the codes from `Data_before_processing` (Bond Yields and `NIFTY_CE.csv` and `NIFTY_PE.csv`) from here. Add them in the folder containing the codes.

2. Run all the cells in `Data_processing_call.ipynb` notebook for call and `Data_processing_put.ipynb` for put till the one asking for `new_model_call_impl_sig.csv` or `new_model_put_impl_sig.csv`. At this point, run the Matlab code `bsm_sig_call.m` or `bsm_sig_put.m` respectively.

3. Once done, continue running the rest of the code.

4. In the end, you should have produced `new_model_final_call_df.csv` or `new_model_final_put_df.csv` which will be used as inputs for modelling part explained just above.

If there are any issues accessing the datasets or running the code please let us know so that we can rectify the issue.
The notebooks are quite bulky to run. Hence, we have left the outputs from our own execution as it is for ease of analysis.
We recommend using Colab notebook links (attached below) with settings-
Runtime $\rightarrow$ Change Runtime $\rightarrow$ GPU

10 Important Links

1. One Drive Link for Datasets
2. Google Drive Link for Datasets
3. Overleaf Link for Latex Document of the Report
4. Call Codes - Colab for Data Processing of Call Matlab for Implied Volatility Colab for Call Modelling
5. Put Codes - Colab for Data Processing of Put Matlab for Implied Volatility Colab for Put Modelling

References

[1] S. Sehgal and N. Vijayakumar, "Determinants of Implied Volatility Function on the Nifty Index Options Market: Evidence from India," *Asian Academy of Management Journal of Accounting and Finance (AAMJAF)*, vol. 4, no. 1, pp. 45–69, 2008.

[2] "Nse historical options." `https://www1.nseindia.com/products/content/derivatives/equities/historical_fo.htm?`

[3] R.Lundmark, "Forecasting call option prices: A quantitative study in financial economics."

[4] "3-month bond yield for indian market." `https://in.investing.com/rates-bonds/india-3-month-bond-yield?desktop=1`.

[5] A. Al-Aradi, A. Correia, D. Naiff, G. Jardim, and Y. Saporito, "Solving nonlinear and high-dimensional partial differential equations via deep learning," 2018.

[6] S. Liu, C. Oosterlee, and S. Bohte, "Pricing options and computing implied volatilities using neural networks," *Risks*, vol. 7, p. 16, Feb 2019.

[7] J. Ruf and W. Wang, "Neural networks for option pricing and hedging: a literature review," 2020.

[8] S. Kanojia and N. Jain *IOSR Journal of Business and Management (IOSR-JBM)*, vol. 19, pp. 01–08, July 2017.