Neural Network for Option Pricing [*]

Congcong MA Sen ZHU

July 1, 2023

Abstract

This term paper studies the application of neural network models to option pricing. Specifically, we build two neural network models for computing the Black-Sholes implied volatility and Heston call option price. The usage of neural network is able to speed up the implied volatility computation by 10 times and heston call price by 8000 times, with the cost of accuracy. On average, the mean absolute error for the implied volatility model is 3% and the mean absolute percentage error for the heston call option price is 10%. Future possible improvements are also pointed out.

Keywords: Neural Network, Option Pricing, Implied Volatility

[*]This is the term paper of Econ 678 at Duke University, codes are avaliable at https://github.com/Donis666/NNOptionPricing

1 Introduction

This research explores the application of neural network models under the context of option valuation. Neural network models are the basic elements of deep learning. Nowadays their applications ranges from self-driving cars, language models (such as ChatGPT) to Recommendation Systems. Briefly, neural networks (NN) are a type of machine learning models known for their predictive accuracy. By the Universal Approximation Theorem, neural network models are able to theoretically approximate any (Borel measurable) functions. However, the cost of high accuracy is often interpretability. Unlike easily interpret-able models like linear regression, the fitted neural network models do not offer much insights on how prediction is actually given. Therefore, neutral network models are mostly used in areas where predictive accuracy is way more important than model interpret-ability, such as computer vision and natural language processing.

The major results of this research project are two neural network models for computing Black-Scholes implied volatility and Heston call option price. For the implied volatility model, we achieve a mean absolute error of 3%. For the Heston call price model, we achieve a mean absolute percentage error of 10%. Therefore, if such error rate is acceptable, we are able to compute the implied volatility and heston call price significantly faster than the original iterative or simulation methods. Our paper also provides suggestions on fitting neural network models to the data of option prices.

2 Neural network for implied volatility

2.1 Motivation of the volatility model

This section trains a neural network model to predict the implied volatility of call options by the Black-Scholes model. The Black-Scholes model is a revolutionized mathematical model for the dynamics of derivative investments in financial markets. For simplicity, we assume constant volatility in our Black-Scholes model as opposed to stochastic volatility in more complicated models which we will discuss later. Here are some classical assumptions of the Black-Scholes model:

- Interest rate is known and constant through time
- The stock follows a random walk in continuous time and the variance of the stock price follows a log-normal distribution
- Volatility is constant
- There are no dividends during the lifetime of the stock
- Only European options are traded
- No transaction cost
- Fractional trading is possible

The one parameter in the Black-Shcoles pricing formula that cannot be directly observed is the volatility of the stock price. In practice, traders usually work with implied volatility which is the volatility that, when used in conjunction with the Black-Scholes option pricing formula, gives the market price of the option. As a result, Black-Scholes model is often used to speculate implied volatility given option prices and not the other way around.

Though there exist analytical solution equations for implied volatility, we often rely on computers to compute implied volatility using numerical methods. One of the numerical methods that are commonly used in the industry is the iterative method. However, in our project, we apply deep neural networks as powerful high-dimensional functional to approximate the multidimensional pricing function from model parameters to implied volatility.

2.2 Training of the volatility model

To test how accurately the deep neural networks can predict implied volatility of call options by learning the Black-Scholes equation, we first need to simulate a set of call option prices using a set of parameters shown in the table below.

Table 1: Simulation details

Parameters	Simulation Range
$N(number of samples)$	300,000
$Stockprice(S)$	[10,500]
$Maturity(T)$	$[\frac{1}{365}, 3]$
$Riskfreerate(r)$	[0.01, 0.03]
$Volatility(\sigma)$	[0.05, 0.9]

There is one more variable, strike price (K) required to calculate option prices but not included in the Table above. The reason why we do not randomly generate strike prices is due to its bad prediction performance. Therefore, we generate the strike prices by $K_i = S_i e^{q_i t}$, where q is a random number ranges from -0.03 to 0.03. By doing so, we can ensure that the strike price does not deviate from the stock price to much.

After we have simulated required parameters, we then need to specify the build-up of the neural networks. We first separate the dataset incorporating parameters and simulated call prices into two subsets: training set (90%) and testing set (10%). The training set is used for in-sample model training while the testing set is used for comparing predicting out-of-sample results generated by the training set.

As a baseline illustration, we train a model with 4 hidden layers with neurons (200, 300, 300, 200) and 5 activation functions (leakyRelu, Elu, Elu, Elu, Relu). One thing worth noting is that, by default, we apply batch normalization for faster and more stable training of neural networks. The optimizer is set to be 0.005 and the loss function is MAE (Mean absolute Error). The train and validation errors are plotted in the figure below.

This is a good demonstration of training and validation errors for that the validation errors decrease rapidly after just no more than 10 epochs. As the number of epoch increases

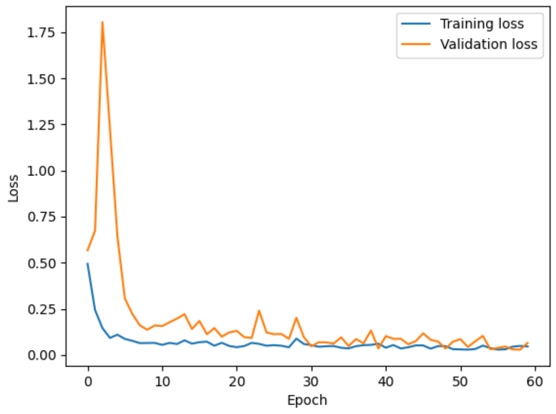


Figure 1: Training and Validation Error of our volatility model

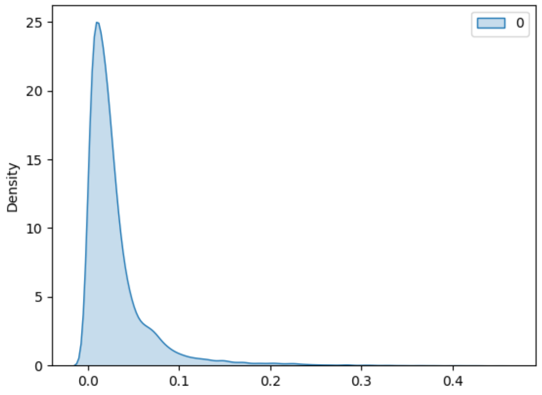


Figure 2: Density of prediction errors (Mean Absolute Difference)

to more than 10, the validation errors decrease steadily and get closer and closer to the level of training errors which remains around 10%.

One remark here is that we intentionally did not include too many epochs for training neural networks. We did this because we anticipate quick drops of validation errors since the implied volatility is calculated by call option prices with known function (i.e. the Black-Scholes formula) and such result validates the applicability of the our neural networks.

2.3 Analysis of the testing errors

Figure 2 plots the error distribution of our models in the testing sample. Even though the maximum error is up to 40%, most of he errors are clustered close to 0. The mean of errors is 0.030612 and the 75% percentile of errors is 0.035139.

Small errors also translate into faster computational speed. Traditional methods view computing implied volatility as a minimization problem. One possible form of minimization is minimizing the absolute difference between the market and Black-Shcholes price. Let V below be the value of a European call or put option.

$$\sigma^* = \arg \min_{\sigma \in (0,6]} |V_{market} - V_{BS}(S, K, T, r, \sigma)|$$

Such minimization is computed by Scipy's minimize scalar function using Brent's methods. However, the time taken for this method to generate implied volatility is around **42 seconds**. This is significantly slower than the time taken for our neural networks (around **4.2 seconds**) to compute the implied volatility given the same data set.[1]

3 Neural network for the Heston model

3.1 The Heston Model and its simulation

This section trains a neural network model to predict the price computed by Heston model. Different from Black-Scholes, Heston model assumes stochastic volatility. Incorporating stochastic volatility provides explanation for certain empirical observations such as volatility smile and the inverse relationship between asset returns and the volatility. Briefly, the Heston model assumes that the stock price process satisfies

[1]Based on the M1 Pro chip Macbook Pro

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^S$$

$$dv_t = \kappa(\theta - v_t)dt + \xi\sqrt{v_t} dW_t^v$$

where dW_t^S and dW_t^S are Wiener processes with correlation ρ. μ is the drift term of the stock return, θ is the long run average variance, κ is the rate of mean reversion and ξ is the volatility of v_t.

Following this dynamics, the option price could be computed using risk-neural pricing. In other words, if C denotes the current option price at time 0, with maturity T, constant risk free

$$C = \mathbb{E}^Q(e^{-rT}\max\{S_T - K, 0\})$$

where Q is an equivalent measure such that the discounted process $e^{-rt}S_t$ is a martingale. In practice, the option price could then be computed through Fourier Transform or simulation. In this project, we obtain the option price by simulating the following process:

$$S_{t+i} = S_t e^{r - \frac{v_t}{2}dt + \sqrt{v_i}dW^S}$$

$$v_{t+1} = v_t + \kappa(\theta - v_t)dt + \xi\sqrt{v_t}dW_t^V$$

Notice that the drift term of S_t becomes r because the process is simulated under the measure Q such that $e^{-rt}S_t$ is a martingale. With the process S_T simulated, the option price is computed by

$$C = \sum_{n=1}^{N} \frac{1}{N}\max\{S_T - K, 0\}$$

3.2 Model Training

As before, the training data is generated by simulation. Specifically, we generate the training data set by randomly select the model parameters and the compute the option price with the method mentioned above. The detailed information if given in table 2.

Table 2: Simulation details

parameters	Simulation Range
N (number of samples)	20,000
S	[10, 500]
T	$[\frac{1}{12}, 3]$
r	[0.01, 0.1]
v_0	[0.05, 0.9]
θ	[0.01, 0.8]
κ	[0, 10]
ρ	[-0.99, 0.99]

There are two more parameters not mentioned in Table2, namely, the strike price K and the vol of vol ξ. Learning from our previous lessons, we generate the strike price K by $K_i = S_i e^{q_i t}$, where q is a random number ranges from -0.2 to 0.2. This practice ensures that the strike price does not deviate from the stock price too much. Secondly, ξ is generated such that $\xi^2 < 2\theta\kappa$ to ensure that the volatility v_t is strictly positive.

With these parameters set, we simulate S_T by dividing T into 2000 steps and running the 20, 000 different paths, according to the above formula. In our computer [2], it takes 2.3 seconds for each price computation and it takes **13 hours** to generate the entire 20,000 data points.

Compared to previous models, training a neural network to fit the Heston model is significantly more difficult. First, the function mapping from parameters $(S, K, T, r, v_0, \theta, \kappa, \rho, \xi)$ to the final call price is much more complex than the Black-Scholes, which motivates larger network size. However, due to time limit, the sample size ($N = 20,000$) is much less than our previous models ($N = 300,000$). Third, since our model relies on approximating the functional relationship from data, the simulation error when computing the call price also increases the difficulty of training.

As a baseline illustration, we train a model with 6 hidden layers with neurons (100, 200, 200, 300, 200, 200) and activation function (leakyRelu, Elu, Elu, Elu, Elu, Elu, relu). The

[2] M1 Pro chip macbook Pro

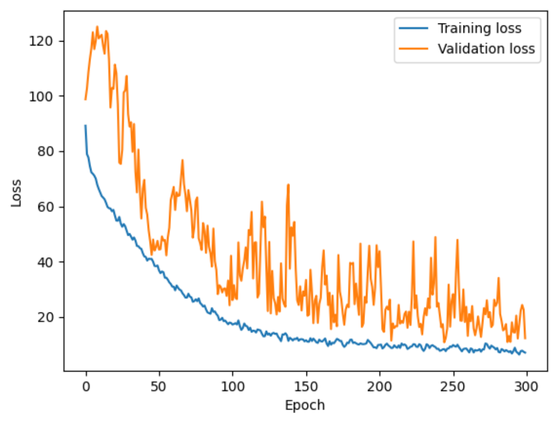


Figure 3: Training and Validation errors of baseline model

optimizer is set to be 0.001 and the loss function is MAPE (Mean Absolute Percentage Error). The train and validation errors are plotted in the following Figure 3.

Two observations are worth noticing. First, as the number of epoch increases to more than 100, the training error decreases very slowly, while the validation error remains at around 20%. This is an indication of over-fitting. Second, the validation loss fluctuates greatly. One possible explanation is that we have chosen an inappropriate learning rate so that the learning algorithm jumps between various local minimums.

We partly mitigate these difficulties by adding dropout layers and decreasing the value of learning rate. Dropout is a regularization technique used in deep learning to prevent overfitting of the neural network model. The basic idea behind dropout is to randomly drop out (i.e., set to zero) some of the nodes in the neural network during training. This has the effect of reducing the co-adaptation between neurons, forcing them to learn more robust features that generalize better to new data. The following Figure 4 demonstrates the effect of adding a dropout of 0.5 (randomly drop out 50% 0f the nodes) to our previous model. The validation loss now has a better tendency of going down with the training error.

The second trick we use is to reduce the learning rate and simultaneously increase the number of training epoch, in hope of pursuing a more stable decrease of validation loss. The following Figure 4 demonstrates the effects of decreasing learning rates. Note that even though it helps stabilize the validation error, it takes significantly larger training epoch and hence more time.

3.3 Analysis of the Testing Error

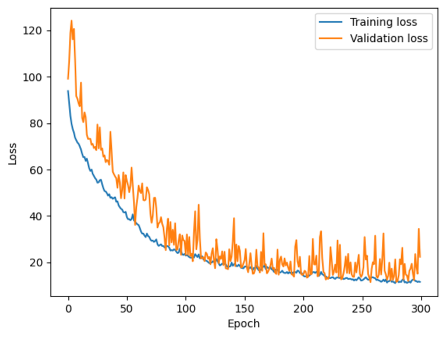
The final model we choose for further analysis uses learning rate of 0.0008 and dropout ratio of 0.3 for each layer. This model achieves a mape of **9%** and it takes only **0.5 seconds** to compute 2000 call prices of our test sample. We choose this model not because it is the best we have achieved during our experiment, but it is the most consistent and reproducible results we have observed during all fine-tuned models[3]. This result indicates that if we could tolerate an average of 10% errors, the neural network model is able to compute the a large number of prices **8000 times** faster.

Even though 10% might seem intolerable for many financial application, the model actually still has lots of space for improvement. First, when time permits, it should be helpful to generate more data for training. Note that our model has more than 100,000 parameters while the number of samples is only 20,000. Although this is not an uncommon practice in the field of deep learning, increasing the number of samples should be beneficial. Second, one could take more accurate methods to generate the Heston option price, such as Fourier methods or better simulation techniques. Third, this model only utilizes very simple network structures. More complicated structures such as convolution neural network or residual neural network might worth trying to improve the accuracy of prediction.

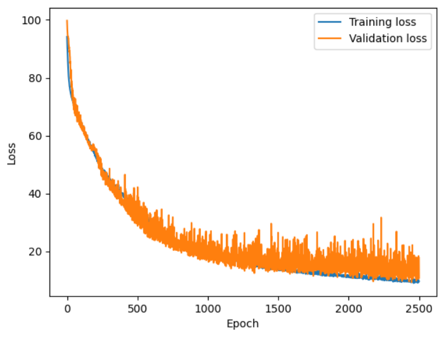
4 Conclusion

To summarize, this paper applies neural network models to compute option's implied volatility and price. Deep learning techniques enable faster computation, with the cost of accuracy. Our model's average error rate is 3% and 10% for volatility and price respectively. Future re-

[3]The best model we trained achieve a mape of 5%



(a)



(b)

Figure 4: (a) Adding dropout (b) Decreasing learning rate

search could focus on either training networks for other option models such as jump diffusion or using the trained network for model calibration, greek computation etc.