Learning Price Function Under No Arbitrage

Samira Amiriyan [1]

[1]Department of Mathematical Sciences, University of Liverpool, UK

Introduction

In 1973, for the first time, Fischer Black and Myron Scholes published their famous article [1] and proposed to model the underlying asset with SDEs for derivatives pricing. They assumed the underlying process, S_t at time t, satisfies the following stochastic differential equation, known as the Black–Scholes SDE:

$$dS_t = rS_t dt + \sigma S_t dW_t,$$

where r , σ, and W_t are a continuous risk-free interest rate, the instantaneous volatility, and Brownian motion respectively. The Black and Scholes call option pricing formula with Strike K and expiration time T on the underlying asset S_t is as follows:

$$C(S_t, K, t, \sigma) = S_t.N(d_1) - Ke^{-r(T-t)}.N(d_2),$$

where $d_1 = \frac{ln(\frac{S_t}{K}) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}$ and $d_2 = d_1 - \sigma\sqrt{T-t}$.

Advantages:

1. It provides a closed-form formula for option pricing, making it easily applicable in the industry.
2. It satisfies the derivatives conditions for the absence of arbitrage.

Disadvantages:

1. The constant assumption of volatility within this model.
2. It fails to capture some features of the market such as Severe Price jumps, Implied volatility, and so on.

Although other models such as Local Volatility models, Stochastic Volatility models, Rough Volatility models have been introduced to address the issues with Black-Scholes, numerical pricing with these alternative models is computationally intensive and difficult to calibrate, which is why transitioning to a machine learning approach is preferable. In 1993, Malliaris and Salchenberger [2] used Artificial Neural Network (ANN) for option pricing in their paper. After this article, more than a hundred articles have been published in this field.
Here, we will utilize Artificial Neural Networks to learn the Call price function C = C(T,K) based on empirical data, independent of traditional models, with No-arbitrage conditions.

Artificial Neural Network

An Artificial Neural Network (ANN) is a computational model inspired by the way biological neural networks in the human brain process information. The architecture of ANN includes:

1. Neurons (Nodes): The basic unit of a neural network is the neuron. Neurons are organized in layers and are connected by weighted links.
2. Layers:
   Input Layer: The input layer receives the initial data (Inputs).
   Hidden Layers: These layers perform most of the computations required by the network.
   Output Layer: The output layer produces the final result of the network.
3. Weights: Each connection between neurons in different layers has an associated weight, which is adjusted during the training process to minimize error.
4. Activation Function: Determines the output of a neuron based on its input. It introduces non-linearity into the model, enabling the network to learn complex patterns.

Figure 1 shows a NN with 4 inputs and 2 outputs.

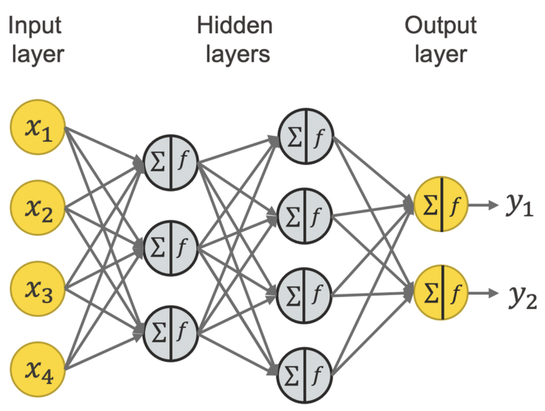

Figure 1. Artificial Neural Network

According to the Universal Approximation Theorem [3], any neural network with a single hidden layer and a non-polynomial activation function can approximate suitable function $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$.
As, each option price is a mapping like $C : (T_i, K_j) \longrightarrow C(T_i, K_j)$ so we can learn this map with ANN.

Learning Price Function under the No-Arbitrage Conditions

It is known from the option pricing theory, that the necessary and sufficient conditions for the Call option prices $C = C(T, M)$, where $M = \frac{S_t}{K}$ is "moneyness", to be arbitrage-free are:

$$\frac{\partial C}{\partial T} \geq 0 \quad , \quad \frac{\partial C}{\partial M} \geq 0 \quad , \quad \frac{\partial^2 C}{\partial M^2} \geq 0.$$

Charles Dugas et al. [4] demonstrated that a neural network with the following architecture is a universal approximator for the option pricing function under the no-arbitrage condition.

$$F(X) = e^{w_0} + \sum_{i=1}^{H} e^{w_i} \left(\prod_{j=1}^{c} \zeta \left(b_{ij} + e^{v_{ij}} x_j \right) \right) \left(\prod_{j=c+1}^{n} h \left(b_{ij} + e^{v_{ij}} x_j \right) \right),$$

where, $\zeta(s) = \ln(1 + e^s)$ (softplus activation function), $h(s) = \frac{1}{1+e^{-s}}$ (sigmoid activation function).
It is easily seen that the output is always positive, the first derivatives with respect to x_j variables are positive, and the second derivatives with respect to the x_j for $j \leq c$ variables is also always positive.
This architecture has several issues. For example:

1. The derivatives with respect to K (strike price) and T (time to maturity) also include derivatives with respect to σ (volatility), which is not intended.
2. The model does not accurately reflect how the price behaves for very large values of K and T.

Numerical Results

Our dataset consists of Apple (AAPL) call options with expiration dates ranging from June 28, 2024, to December 18, 2026, and various strike prices.
The inputs to the neural network are pairs (T, M), and the output is $\frac{C}{K}$.
20% of the data is used as the test set, and 80% is used for the training set.
Additionally, $H = 5$ represents the number of hidden units, $c = 1$ is the number of inputs for the softplus function, and $n = 2$ is the total number of inputs.
We used the MSE loss function which measures the difference between the network's prediction and the actual target values

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

where y_i is the real value, $\hat{y}_i$ is output of the NN, and n is the number of outputs.
We trained it for up to 1000 epochs, and the results can be seen in the following figures.

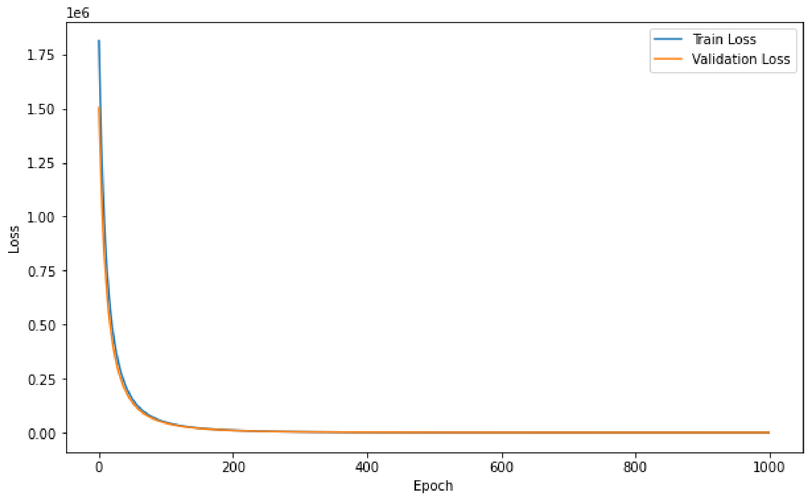

Figure 2. Training and Validation Loss Curves Over Epochs

In the figure 2, you can see that both the training and validation loss curves decrease steadily and approach zero over time. This means the model is learning effectively from the data. The fact that the curves are closely aligned shows that the model generalizes well, performing consistently on both the training and new, unseen validation data. This suggests that the model has the right level of complexity—it's learned the data patterns without overfitting. Overall, these results are promising and indicate that the model is likely to perform well on new data it hasn't seen before.

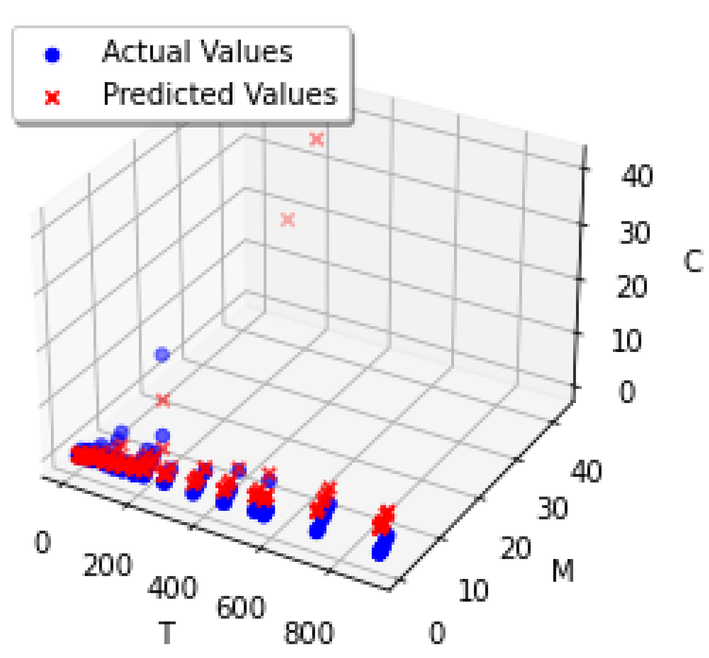

Figure 3. True Values vs. Predicted Values

In the figure 3, most of the points are very close, showing that the model's predictions match the actual values quite well. This tight clustering indicates that the model is making accurate predictions and effectively capturing the patterns in the data.
To view the Python code, please scan the QR code below.

SCAN ME

Ongoing Research

We aim to incorporate volatility (σ) as an input alongside T (time to maturity) and K (strike price) into the model to address the issues we mentioned. By including volatility as an explicit input, we seek to resolve these problems and improve the model's accuracy.

References

[1] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of political economy*, vol. 81, no. 3, pp. 637–654, 1973.

[2] M. Malliaris and L. Salchenberger, "A neural network model for estimating option prices," *Applied Intelligence*, vol. 3, pp. 193–206, 1993.

[3] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[4] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating functional knowledge in neural networks," *Journal of Machine Learning Research*, vol. 10, no. 42, pp. 1239–1262, 2009. [Online]. Available: http://jmlr.org/papers/v10/dugas09a.html

samira.amiriyan@liverpool.ac.uk