Machine Learning Methods for Vanilla Option Pricing and Heston Calibration

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
DEPARTMENT OF FINANCIAL ENGINEERING
May 4, 2023

Contents

Executive Summary

This report details the application of machine learning methods to vanilla option pricing and the Heston calibration. A stochastic volatility model, such as the Heston model, is a widely used option pricing model, but it can be computationally expensive and difficult to calibrate. However, machine learning (ML) methods have shown great potential in providing fast and accurate solutions to complex problems, making them a potential alternative for option pricing and calibration.

We first generate European and American options using the Black-Scholes pricer and train different machine learning models to price them. We then proceed to process option prices using the Heston model, using the machine learning models to learn the mapping between input features and option prices. Finally, we use an optimizer called differential evolution (DE) as a pricer for parameter calibration and mean absolute error (MAE) to select the best model.

We found that using machine learning methods for option pricing and calibration has the potential to lead to more accurate and efficient solutions that improve the overall performance of financial models. However, in the context of financial modeling, machine learning approaches have limitations and potential pitfalls that require further exploration.

1. Introduction

Option pricing is a critical task in financial modeling, and the accuracy and efficiency of pricing methods have significant implications for financial decision-making. The stochastic volatility model, such as the Heston model, is a widely used model for option pricing, but it can be computationally expensive and challenging to calibrate. Recently, machine learning methods, particularly deep neural networks, have shown promise in providing fast and accurate solutions to complex financial problems. Machine learning methods can potentially offer an alternative approach to option pricing and calibration that is both efficient and accurate.

This report explores the application of machine learning methods for vanilla option pricing and Heston calibration. In the first part, we implement different machine learning models, such as decision trees, random forests, and neural networks, to price European/American options using the Black-Scholes pricer (Section 3). We evaluate the performance of these models using metrics such as mean absolute error (MAE) and mean squared error (MSE). In Section 4, we implement a neural network-based framework for Heston calibration. We use machine learning models to learn the mapping between input features and option prices, providing a more efficient way to price options compared to traditional methods.

2. Literature Review

2.1 Accelerated American Option Pricing with Deep Neural Networks

Given the competition in the market-making environment, the ability to quickly quote option prices that are consistent with changing market conditions is critical. Anderson and Ulrych [1] proposed a new approach to accelerate the pricing of American options to near-instantaneous using feed-forward neural networks. The neural network is trained according to a chosen (i.e., Heston) stochastic volatility specification. The findings show that the proposed deeply interpretable pricer leads to a speed-accuracy trade-off compared to typical Monte Carlo or PDE-based pricing methods. Furthermore, the proposed method allows pricing of derivatives with path dependencies and more complex returns, and is suitable for a market-making setting given sufficient computational accuracy and its tractable nature.

2.2 A neural network-based framework for financial model

In this paper, a data-driven approach called CaNN (Calibration Neural Network) is proposed to calibrate financial asset price models using an Artificial Neural Network (ANN). Determining optimal values of the model parameters is formulated as training hidden neurons within a machine learning framework, based on available financial option prices. It presents the framework to calibrate a financial model by means of machine learning. Training the ANNs and calibrating financial models both boil down to optimization problems, which motivates the present machine learning-based approach to model calibration.

2.2.1 The forward pass: learning the solution with ANNs

The forward pass employs an artificial neural network (ANN) to learn the solution generated by different numerical methods and maps the input to the output of interest. The authors merge two ANNs to map the Heston parameters to implied volatility. The forward pass consists of training and prediction, where the network architecture and optimization method must be defined. The authors discuss the trade-off between ANN's computation speed and approximation capacity and various techniques for training ANNs, including gradient-based methods, random initialization, batch normalization, and dropout operation. The authors found that their ANNs model did not

encounter over-fitting, even with a zero-dropout rate, as long as sufficient training data were provided.

2.2.2 Differential Evolution

Differential Evolution (DE) is a population-based derivative-free optimization algorithm that does not require any specific initialization. Using DE, global optima can be found even if the objective function is non-convex. The general form of the DE algorithm usually includes the following four steps: initialization by generating the population, add randomly sampled difference to each individual, determin the mutated candidates that may enter the next evaluation stage and select the candidate with the corresponding target.

3. Methodology

3.1 Random Forest

Random Forest works by creating multiple decision trees and aggregating their results to make a final prediction. Each decision tree is trained on a randomly sampled subset of the training data, and each tree is constructed by randomly selecting a subset of features at each node. By using random sampling and feature selection, Random Forest is able to reduce overfitting and improve the model's generalization performance.

When making a prediction, the Random Forest algorithm aggregates the predictions of all the decision trees to arrive at a final prediction. The most common method for aggregating the results is to take the majority vote (in the case of classification) or the average (in the case of regression) of the predictions made by all the decision trees. The process is as follows.

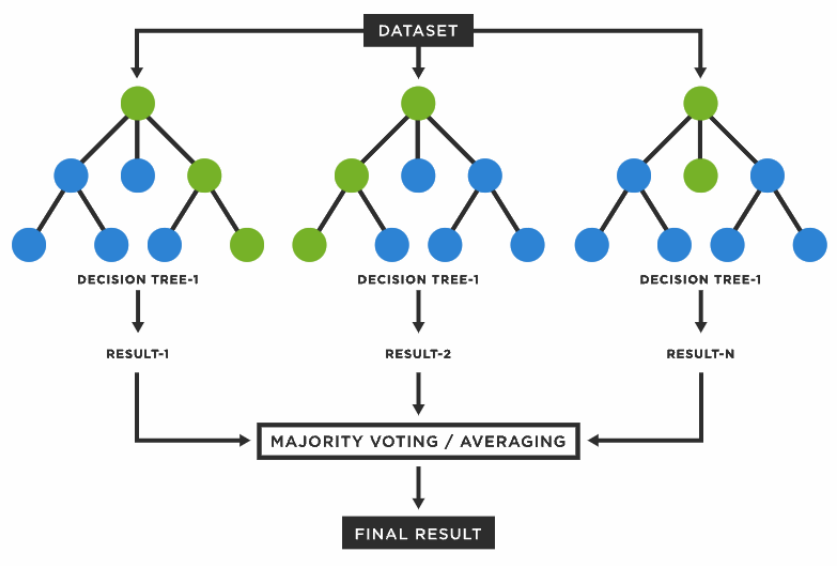


Figure 1 Random Forest Illustration

Random Forest has several advantages over other machine learning methods. It's robust to outliers and noisy data, and it can handle high-dimensional data with many features. It's also relatively easy to use and interpret, and it's not as sensitive to hyperparameter tuning as other ensemble methods like Gradient Boosting.

We utilized the Random Forest approach to handle parameters and determine the option price in our research, as it is particularly beneficial for handling vast and intricate datasets that could pose difficulties for other machine learning techniques.

3.2 XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm that belongs to the family of gradient boosting methods. It is designed to handle both regression and classification problems and has been used extensively in various machine learning competitions and industry applications.

XGBoost works by iteratively adding decision trees to the model while minimizing a user-specified loss function. It starts with a single decision tree and gradually builds an ensemble of trees by adding new trees that improve the overall performance of the model. At each iteration, the algorithm tries to fit a new tree to the errors made by the previous trees, with the aim of reducing the overall error of the model. The process is as follows.

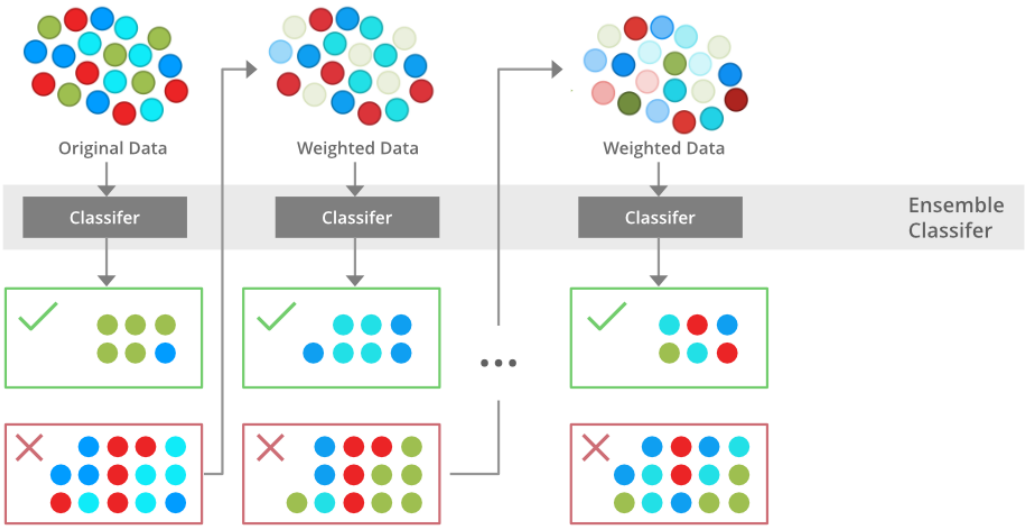


Figure 2 XGBoost Illustration

One of the key strengths of XGBoost is its ability to handle large and complex datasets. It uses a technique called "regularization" to prevent overfitting, which is a common problem in machine learning when a model becomes too complex and starts to fit the training data too closely.

XGBoost also supports parallel processing and can take advantage of multiple CPU cores, making it very fast and efficient.

XGBoost has a wide range of hyperparameters that can be tuned to improve its performance. These include the learning rate, which controls the step size taken during each iteration, the maximum depth of the trees, and the minimum number of samples required to split a node.

XGBoost is a powerful and versatile machine learning algorithm that can be used in a variety of applications. Its ability to handle large datasets, its efficiency, and its flexibility in tuning hyperparameters make it a popular choice among machine learning practitioners. In our research, this model shows its superiority in processing tens of thousands of option data.

3.3 CatBoost

CatBoost is a gradient boosting machine learning algorithm that was developed by the Russian search engine company, Yandex. It's designed to handle datasets with categorical features, which are common in many real-world applications.

CatBoost works by constructing an ensemble of decision trees, similar to other gradient boosting algorithms. However, it incorporates several key innovations that make it particularly effective for handling categorical data. For example, it uses a technique called "ordered boosting," which sorts categorical features by their statistical importance and creates splits based on the order of the categories rather than their values. This allows CatBoost to handle high-cardinality categorical features (i.e., features with a large number of categories) more effectively.

Another key feature of CatBoost is its handling of missing values in categorical features. It's able to handle missing values in a way that doesn't require inputing them, which can often introduce bias into the model. Instead, CatBoost uses a separate category for missing values, which is then treated like any other category during the training process.

Figure 3 CatBoost Illustration

CatBoost also has built-in feature selection capabilities, which can be used to identify and remove irrelevant or redundant features from the dataset. This can help to improve the model's accuracy and reduce its computational complexity.

In our research, we employed CatBoost and assessed its performance on our datasets to determine its efficacy.

3.4 LightGBM

LightGBM (Light Gradient Boosting Machine) is a gradient boosting framework developed by Microsoft. It's designed to handle large and high-dimensional datasets with high efficiency and accuracy.

LightGBM works by constructing an ensemble of decision trees in a similar way to other gradient boosting algorithms. However, it incorporates several key optimizations that make it particularly fast and efficient. For example, it uses a technique called "gradient-based one-side sampling" (GOSS) to select only a subset of data instances that have a large gradient value, which can significantly reduce the amount of data needed for training. It also uses a technique called "exclusive feature bundling" (EFB) to group features together based on their statistical similarity, which can reduce the number of features and improve the accuracy of the model.

Another key feature of LightGBM is its ability to handle categorical features efficiently. It uses a technique called "categorical feature optimization" (CFO) to transform categorical features into numerical values that can be used in the training process. This allows LightGBM to handle datasets with a large number of categorical features more efficiently than other gradient boosting algorithms.

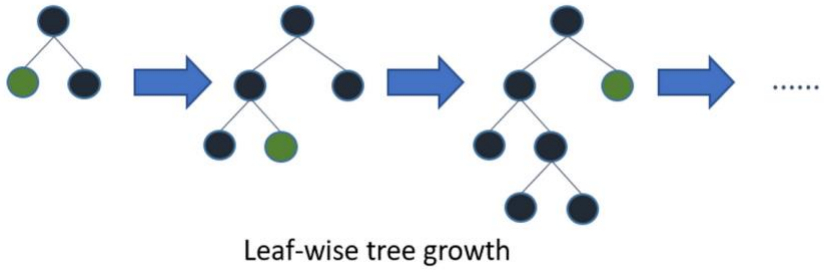


Figure 4 LightGBM Illustration

LightGBM also has several hyperparameters that can be tuned to improve its performance, including the number of trees in the ensemble, the learning rate, and the maximum depth of the trees.

LightGBM is a robust and effective gradient boosting framework that has the potential to cater to diverse machine learning tasks. Its competence in managing large and high-dimensional datasets, employment of optimization techniques, and proficient handling of categorical features have made it a favored option for machine learning professionals. Due to the numerous parameters involved in our model, we utilized LightGBM to derive our outcome.

3.5 Artificial Neural Network

The artificial neural network (ANN) model has been heavily researched in both industry and academia, and there are already many papers that explain its mechanisms in great detail. Therefore, we decided to use a tried and tested ANN model as a starting point for our analysis. We customized the model to suit our specific dataset and to see how well it could predict option prices. By using this approach, we could leverage the existing research and expertise in the field of ANN modeling, while still tailoring the model to our specific needs. This allowed us to get a sense of the predictive power of the ANN model for option pricing, without having to spend too much time and resources on model development.

4. Data

4.1 Black Scholes

The Black-Scholes formula is a mathematical model that is widely used in finance to calculate the theoretical price of a European call or put option, assuming that the underlying asset follows a geometric Brownian motion with constant volatility. The formula was developed by Fischer Black and Myron Scholes in 1973, and Robert Merton made significant contributions to the model's development as well.

The Black-Scholes formula takes into account several factors that influence the price of an option, including the current stock price, the strike price of the option, the time to expiration, the risk-free interest rate, and the volatility of the underlying asset.

The Black-Scholes formula is a powerful tool for pricing options, but it has some limitations. In particular, the formula assumes that the underlying asset follows a geometric Brownian motion, which may not accurately reflect the behavior of real-world financial assets. Additionally, the formula assumes that the risk-free interest rate is constant and that the volatility of the underlying asset is known and constant over the life of the option, which may not be the case in practice. Nevertheless, the Black-Scholes formula remains a widely used and influential model in the field of finance.

In this case we use Black-Scholes formula to price European option used for training the machine learning pricer.

4.2 Heston with Fast Fourier Transform

The Heston model is a popular stochastic volatility model used in finance to price derivatives, such as options, by incorporating volatility risk. The model was proposed by Steven Heston in 1993 and has become a standard model in the finance industry due to its ability to capture the skewness and kurtosis of the implied volatility surface.

Fast Fourier Transform (FFT) is a numerical method used to efficiently compute the Fourier transform of a function. In finance, FFT can be used to price options based on the Heston model, which requires solving a complex partial differential equation (PDE) to obtain the option price. By using FFT, the computational time can be reduced significantly compared to other numerical methods, such as finite difference methods.

The basic idea behind using FFT to price options under the Heston model is to transform the PDE into a form that can be solved more efficiently using FFT. Specifically, the Heston PDE can be transformed into a set of linear equations in the Fourier space. Once the transformed equation is obtained, the option price can be computed by solving the linear equations using FFT.

One advantage of using FFT to price options under the Heston model is that it can be faster and more accurate than other numerical methods. However, it also requires some familiarity with

both the Heston model and FFT. Additionally, the method may not work well for certain types of options or parameter values of the Heston model.

In this case we use Black-Scholes formula to price European option used for training the machine learning pricer and Heston Calibraion.

4.3 Heston with PDE methods

Partial differential equation (PDE) methods are numerical methods used to solve the Heston PDE to obtain the option price. These methods discretize the Heston PDE into a finite number of points on a numerical grid and use finite difference methods or finite element methods to solve the resulting system of equations.

The basic idea behind using PDE methods to price options under the Heston model is to discretize the Heston PDE into a numerical grid, where the option price is defined at each grid point. Once the discretized PDE is obtained, the option price can be obtained by solving the system of equations using numerical methods.

One advantage of using PDE methods to price options under the Heston model is that they can be applied to a wide range of options and parameter values of the Heston model. However, they can be computationally intensive and require a good understanding of numerical methods and the Heston model. Additionally, the accuracy of the method may be limited by the size of the numerical grid used to discretize the Heston PDE.

In this case we use Black-Scholes formula to price European option used for training the machine learning pricer and Heston Calibraion.

4.4 Crank-Nicolson Finite Difference

Finite difference methods (FDM) are numerical methods used to approximate solutions to partial differential equations (PDEs) that describe the behavior of financial instruments. One application of FDM is option pricing. In particular, Crank-Nicolson finite difference (CNFD) is a popular method for pricing options.

The Crank-Nicolson finite difference method is a numerical method that discretizes the PDE describing the option price as a function of the underlying asset price and time. The discretized PDE is then approximated by a system of linear equations, which can be solved iteratively using standard linear algebra techniques. The CNFD method has the advantage of being second-order accurate in both time and space, which means that it can produce more accurate results than simpler methods like the forward difference method.

The Crank-Nicolson finite difference method is a powerful tool for pricing options, but it does require some care in its implementation. The choice of grid spacing and time step size can

significantly impact the accuracy and stability of the solution. Additionally, the iterative solver used to solve the linear system can affect the convergence rate and accuracy of the solution. Nevertheless, with appropriate choices of these parameters, the CNFD method can produce accurate results for a wide range of option types and market conditions.

In this case we use Crank-Nicolson finite difference to price American option used for training the machine learning pricer.

5. Machine Learning Pricer

All models' performances show little difference for European/American Option on simulated BS/Heston datapoints. Therefore, we only present the result on American Option dataset with simulated Heston price as market price for simplicity.

5.1 Random Forest Pricer

Model Structure

While developing our Random Forest model, we concentrated on two primary parameters, namely the maximum features and the number of estimators. We fine-tuned these parameters until the risk of overfitting became minimal, and the mean squared error (MSE) decreased to a reasonable level. The parameters we finally used are as follows.

Table 1 Random Forest Parameters

Parameter	Explanation	Value
max_features	The number of features to consider when looking for the best split.	3
n_estimators	The number of trees in the forest.	200

Result

The trained Random Forest model exhibits a Root Mean Squared Error (RMSE) of approximately 0.008, while the Mean Absolute Error (MAE) amounts to approximately 0.005 on the test set. Graphical representation indicates that the Random Forest model has a tendency to underestimate the price of American call options no matter how large the true price is. Additionally, the Mean Absolute Percentage Error (MSPE) demonstrates an inclination towards higher values as the true price approaches zero.

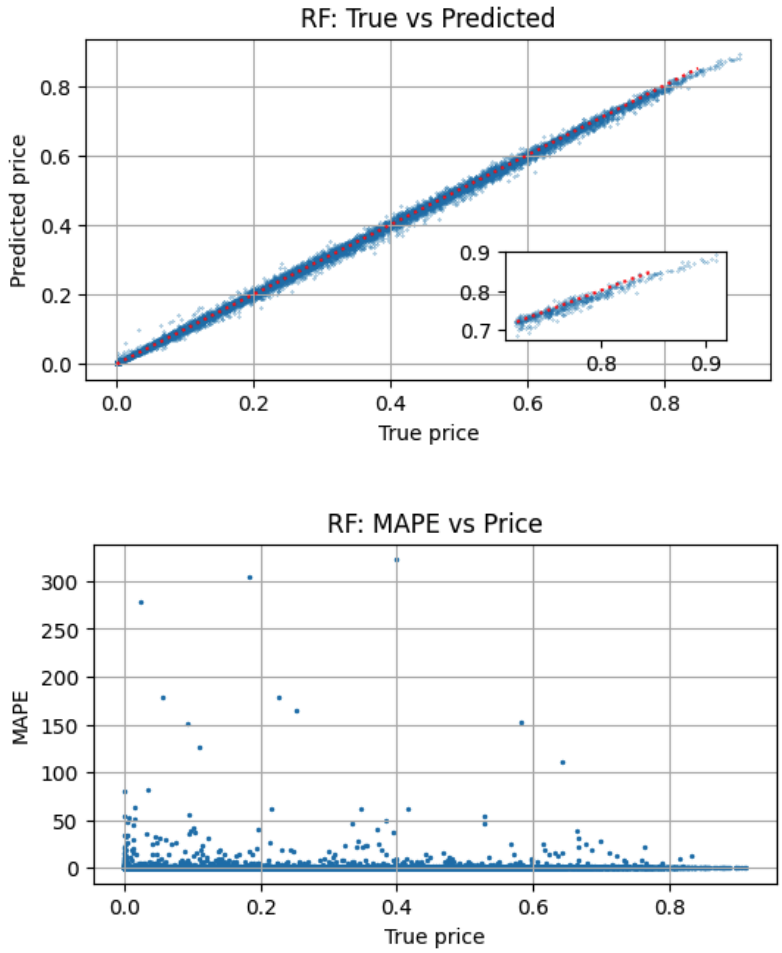


Figure 5 Random Forest Performance

5.2 XGBoost Pricer

Model Structure

During the development of the XGBoost model, we emphasized eight parameters to strike a balance between the model's accuracy and the risk of overfitting.

Table 2 XGBoost Parameters

Parameter	**Explanation**	**Value**
colsample_bytree	It is the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.	0.9
eta	Step size shrinkage used in update to prevents overfitting.	0.05
gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree.	0
reg_alpha	L1 regularization term on weights. Increasing this value will make model more conservative.	0
reg_lambda	L2 regularization term on weights. Increasing this value will make model more conservative.	0
min_child_weight	Minimum sum of instance weight (hessian) needed in a child.	5
num_parallel_tree	Number of parallel trees constructed during each iteration.	3000
max_depth	Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit.	6

Result

The XGBoost model demonstrates an RMSE of 0.005 and an MAE of 0.004. Graphical analysis indicates that the XGBoost model's predicted call option prices closely align with the actual values. Moreover, there is a higher likelihood of encountering large Mean Absolute Percentage Error (MAPE) values when the actual price approaches zero.

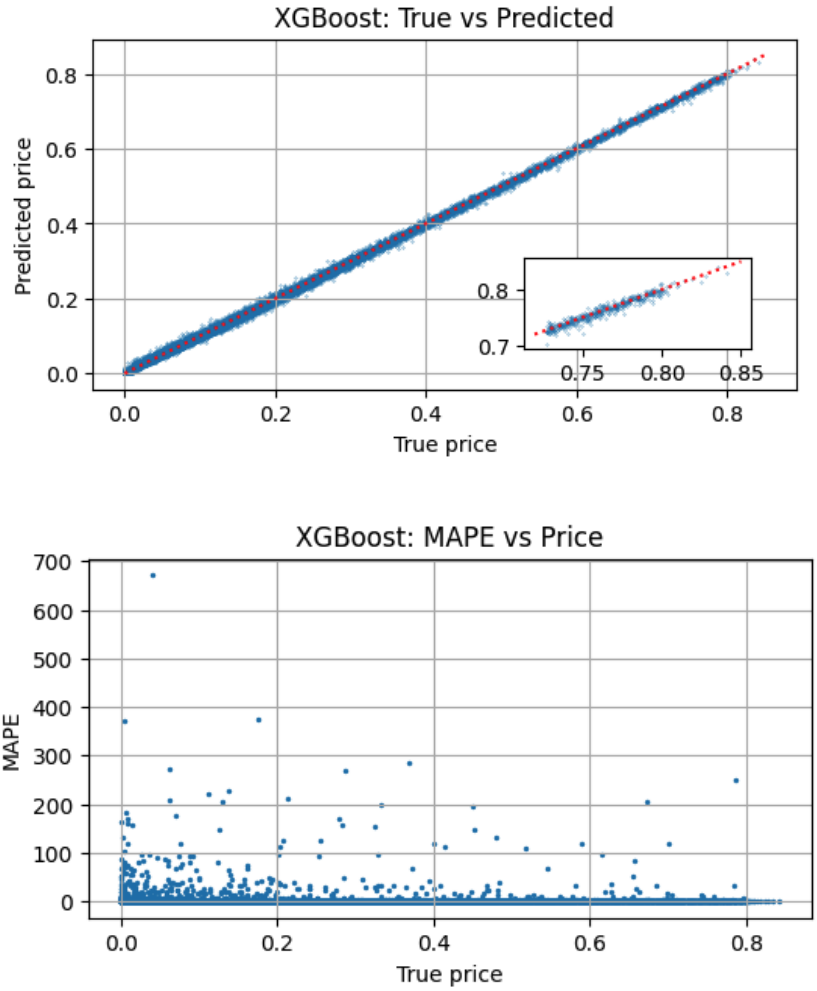


Figure 6 XGBoost Performance

5.3 CatBoost Pricer

Model Structure

We focused on seven parameters when tuning the parameters of CatBoost. The parameters and their meanings are listed as follows.

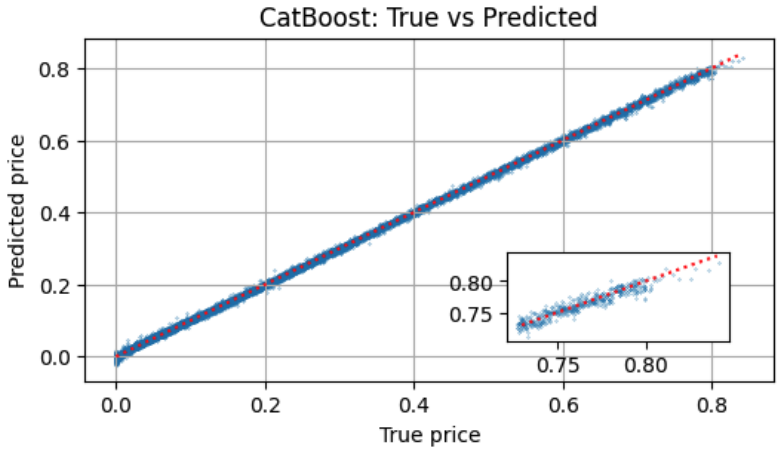
Table 3 CatBoost Parameters

Parameter	Explanation	Value
learning_rate	This setting is used for reducing the gradient step.	0.1

Parameter	Explanation	Value
bagging_temperature	Defines the settings of the Bayesian bootstrap. Use the Bayesian bootstrap to assign random weights to objects.	1.0
l2_leaf_reg	Coefficient at the L2 regularization term of the cost function. Any positive value is allowed.	0
n_estimators	The maximum number of trees that can be built when solving machine learning problems.	6000
max_depth	Depth of the tree.	5
colsample_bylevel	Random subspace method. The percentage of features to use at each split selection, when features are selected over again at random.	0.9
min_child_samples	The minimum number of training samples in a leaf.	5

Result

The CatBoost model displays an RMSE of 0.004 and an MSE of 0.003. According to the graphs, the model tends to underprice the call options as the price of the underlying asset increases. Generally, as the option price rises, the MAPE tends to decrease. However, the chance of experiencing significantly large MAPE values is consistent across different option prices.



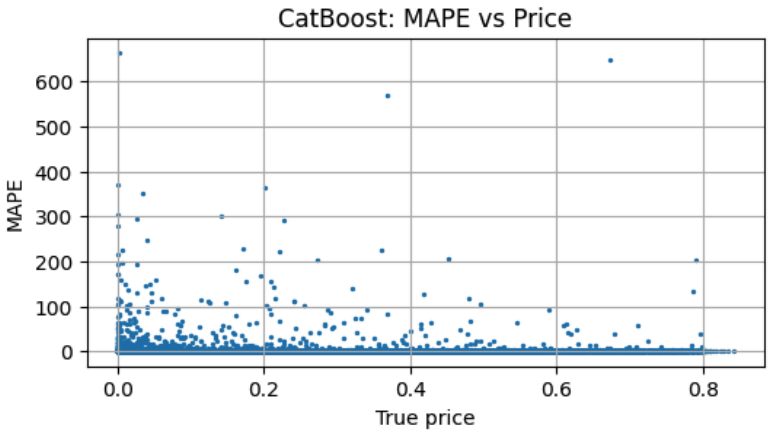


Figure 7 CatBoost Performance

5.4 LightGBM Pricer

Model Structure

During the parameter tuning process for the LightGBM model, we concentrated on eight key parameters to derive an optimal model.

Table 4 LightGBM Parameters

Parameter	Explanation	Value
colsample_bytree	LightGBM will randomly select a subset of features on each iteration (tree) if colsample_bytree is smaller than 1.0.	0.8
learning_rate	The shrinkage rate.	0.05
num_leaves	Max number of leaves in one tree.	30
lambda_l1	L1 regularization.	0
lambda_l2	L2 regularization.	0
min_child_weight	Minimal sum hessian in one leaf.	2
n_estimators	Number of boosting iterations.	6000
max_depth	Limit the max depth for tree model.	8

Result

The LightGBM model has an RMSE of approximately 0.005 and an MAE of 0.004. Analysis of the graphical representation reveals that the LightGBM model has a tendency to underestimate option values as the option price increases. Furthermore, the MAPE demonstrates a tendency to decrease as the option price increases, although there remains a significant probability of encountering significantly large (greater than 100%) MAPE values.

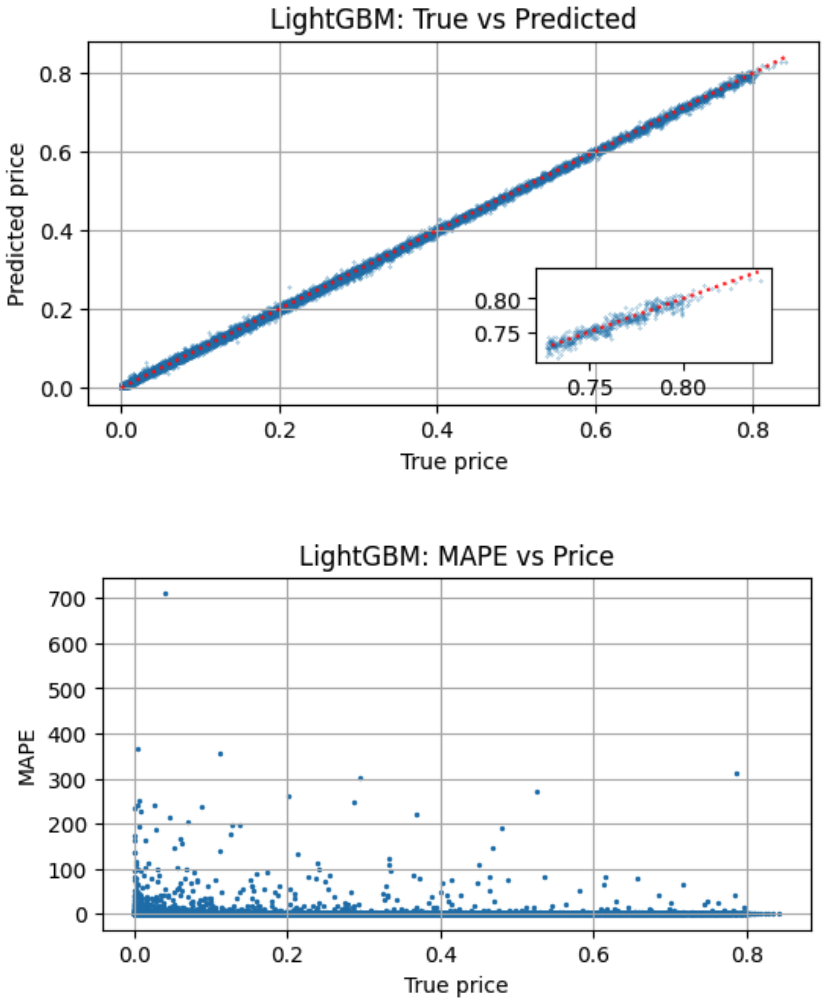


Figure 8 LightGBM Performance

5.4 Neural Network

5.4.1 Neural Network Architecture

The neural network (NN) model we use is a feed-forward MultiLayer Perceptron that employs backpropagation for training. A leaky rectified linear unit (leaky ReLu) activation function is

utilized, and the input layer comprises 9 nodes representing market and Heston model factors. To customize the model's performance, hyperparameters for the hidden layers were adjusted, including custom learning rates for different epochs. The final network structure consists of 1 normalization Layer and 6 hidden layers each with hidden node 64(We consider a linear layer and an activation layer as one layer), has an initial learning rate of 0.001, operates on a dataset size of 100,000, and is trained for 180 epochs over a period of 20 minutes with an early stopping callback (if the test-set losses does not improve for consecutive 5 epoch we will stop training) activated.

Table 5 ANN Model Configuration

Hyperparameters	**config**
$DatasetSize$	100,00
$BatchSize$	128
$InitialLearningRate$	0.001
$ELRA$	0.97
$HiddenLayers(Depth)$	6
$EpochSize$	180
$TrainingTime$	20min

5.4.2 Hyperparameter Optimization

We conducted a series of tests on various combinations of hidden layers and units during the course of our project, using Mean Absolute Error (MAE) and training time as our criterions. Our analysis revealed that a neural network with 6 hidden layers, each containing 64 units, produced the best results. We also experimented with the 4*40-unit combination, as described in the paper "Accelerated American Option Pricing with Deep Neural Networks". However, we found that increasing epoch size did not result in any further reduction in test set losses, and increasing the number of hidden units led to similar results but significantly increased training time. It is worth noting that we did not attempt to increase the depth of the MLP model for either American or European option pricing, as per the guidelines of Multilayer Perceptron Model.

5.4.3 A closer look into Losses

During training, we observed a gradual decrease in the training set loss, indicating that the model was learning and improving its predictive accuracy over time. Meanwhile, the test set loss also decreased over the initial epochs but eventually leveled off, indicating that the model had reached its optimal performance on the test set. Overall, the convergence of loss values for both the training and test sets suggested that our model was capable of learning and generalizing from the data, making accurate predictions on unseen test data. We try different Optimizers including $ReducelrOnPlateau$, $ExponentialLR$ and $StepLR$ to further improvements in our model's predictive performance, and when $epochsize$ is large enough, we see $ExponentialLR$ out perform the other two on MAE. (One thing to note is $ReducelrOnPlateau$ require additional validation set which decrease the size of training set.)

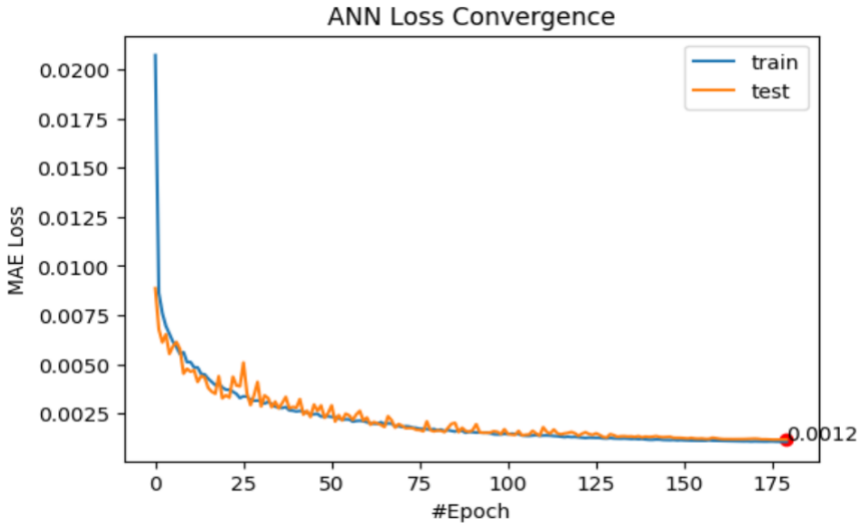


Figure 9 ANN Loss Convergence Plot

By comparing the predicted option price with true price for different $Moneyness$, we could see ANN model perform very well for most of options, with only a few predicted results deviated from true price when the option is Deep in The Money (DITM).

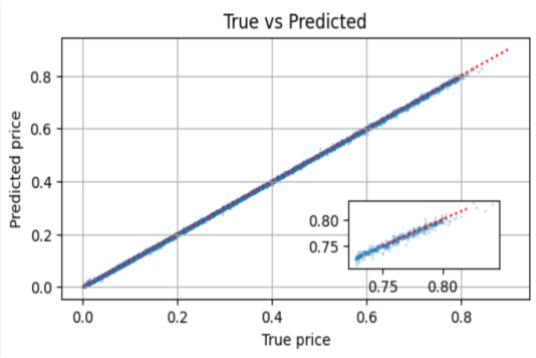


Figure 10 MAPE vs True Price

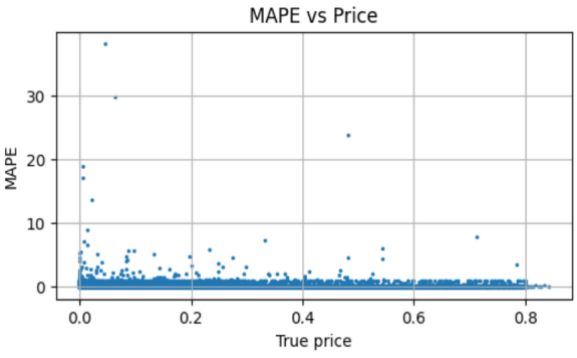


Figure 11 True vs Predicted Price

5.5 Model Comparison

In this analysis, we compare the performance of four different models on two different pricing simulations: Heston model and Black-Scholes model. The four models are artificial neural network (ANN), XGBoost, LightGBM, and CatBoost.

Firstly, we measure the performance of the models using different error quantiles. Our results show that the ANN model outperforms the other models on different error quantiles. This suggests that the ANN model is able to capture the complex patterns in the data better than the other models. However, it's worth noting that the ANN model may require more time and resources to train than the other models.

m	T	r	q	v0	theta	kappa	sigma	rho	AmerCall	XGB price	LGB price	CatB price	ANN price	XGB error	LGB error	CatB error	ANN error
0.2	0.006	0	0	0.01	0.01	0.011	0.01	-0.9	0	-0.038	-0.024	-0.024	-0.004	0	0	0	0
1.391	0.497	0.02	0.012	0.134	0.705	0.713	0.202	-0.457	0.005	0.006	0.006	0.005	0.005	0.001	0.001	0.001	0
2.585	1.004	0.04	0.025	0.257	1.16	1.167	0.41	-0.006	0.068	0.069	0.069	0.069	0.068	0.003	0.003	0.002	0.001
3.783	1.502	0.06	0.037	0.378	1.582	1.591	0.652	0.444	0.231	0.23	0.23	0.23	0.231	0.006	0.005	0.004	0.002
4.948	1.982	0.079	0.049	0.495	1.984	1.984	0.983	0.88	0.755	0.754	0.753	0.756	0.754	0.017	0.014	0.013	0.008
5	2	0.08	0.05	0.5	2	2	1	0.9	0.841	0.84	0.83	0.83	0.83	0.038	0.043	0.03	0.023

Figure 12 Quantile for Both Parameters and Predicted price

When comparing the three tree-based models (XGBoost, LightGBM, and CatBoost), we find that LightGBM and CatBoost have similar performance, while XGBoost performs the worst on the Heston model. This suggests that the LightGBM and CatBoost models are able to handle the non-linear relationships in the Heston model better than XGBoost. However, when the price is simulated using the Black-Scholes model, XGBoost's performance dominates the other two. Despite some variations, three of the models exhibit a similar feature importance pattern to Moneyness being the most significant factor.

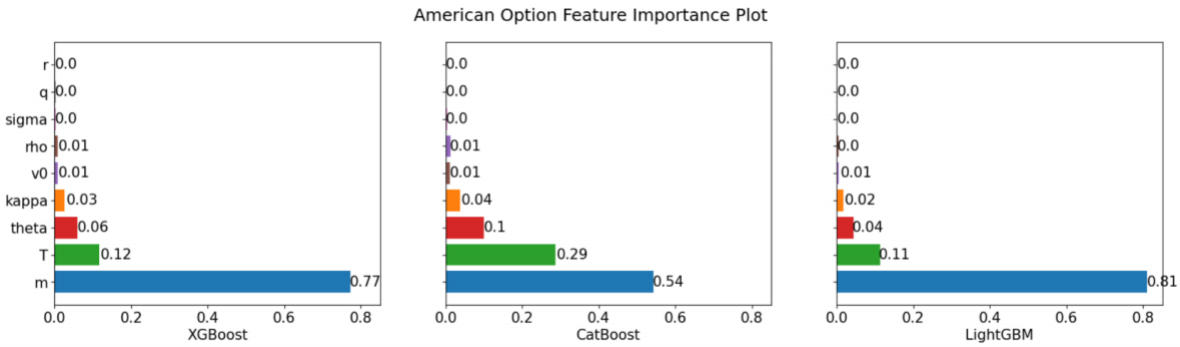


Figure 13 Feature Importance for 3 tree models

We further test the inference time which is the time it takes to predict labels for a test dataset with 5,000 data points, we could see ANN outperform all tree models under the same condition and therefore make it more competitive when it comes to Heston Parameter calibration.

Table 6 Model Inference Time

Model	Time
XGBoost	3.66s
LightGBM	19s
ANN	0.00088s
RandomForest	0.965s

Overall, our results suggest that the ANN model is the best performer in this analysis, while the tree-based models perform differently depending on the nature of the data. It's important to note that these results may not generalize to other datasets or applications, and further experimentation and analysis may be necessary to fully understand the strengths and weaknesses of each model.

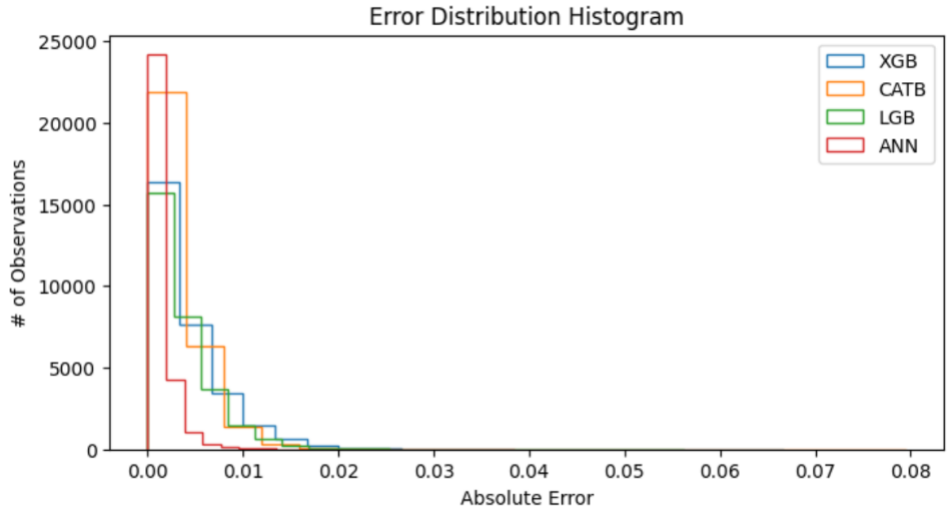


Figure 14 Error Distribution Histogram on American Option

6. Calibration for Heston Parameters

6.1 Outline

Calibrating the Heston model is equivalent to solving the non-linear constrained optimization problem:

$$\min_{\Theta_{\text{Heston}} \in \mathbb{R}^5} F(\Theta_{\text{Heston}}),$$
$$\text{s.t. } l \leq \Theta_{\text{Heston}} \leq u,$$
$$f_{\text{Feller}}(\Theta_{\text{Heston}}) < 0,$$

where the optimization objective function or error measure $F(\Theta_{\text{Heston}})$ defines the distance between the value of the instruments as computed by the Heston model for the parameter set $\Theta_{\text{Heston}}=\{v_0,k,\sigma,\theta,\rho\}$ and the corresponding market values or, as is the case in this paper, generated values. The optimization problem is subject to constraints on the parameters given by the lower (respectively upper) boundaries $l=(0,0,0,0,-1)'$ and $u=(+\infty,+\infty,+\infty,+\infty,1)'$ as well as the Feller condition $f_{\text{Feller}}(\Theta_{\text{Heston}})=\sigma^2-2\kappa\theta<0$.

1, Define the objective function:

$$f(x)=\sum_{i=1}^{n}(C_{\text{market}} - C_{Heston}(x))^2$$

Where $Cmarket$ is the observed market price of the option, $C_{Heston}(x)$is the price of the option predicted by the Heston model using parameter values x, x is a vector of Heston model parameters.

2, Set an initial guess for the Heston parameters

3, Use an optimization algorithm to find the optimal set of parameters. We can use an optimization algorithm, such as the Levenberg-Marquardt algorithm or the Nelder-Mead simplex algorithm, to find the set of Heston parameters that minimize the objective function. This involves iteratively adjusting the parameter values until the objective function is minimized.

4, Evaluate the quality of the calibration. Once the optimal set of parameters is found, we need to evaluate the quality of the calibration. This can be done by comparing the predicted option prices to the observed market prices, calculating statistical measures such as the root-mean-square error or the correlation coefficient, and examining the residuals between the predicted and observed prices.

5, Refine the calibration if necessary. If the calibration is not satisfactory, we can refine it by adjusting the initial guess for the parameters or by changing the optimization algorithm or its parameters. We may also need to consider using different types of options or different maturities in the calibration process.

6.2 XGBoost Calibration

Model Structure

During the development of the XGBoost model, we emphasized eight parameters to strike a balance between the model's accuracy and the risk of overfitting.

Table 7 XGBoost Parameters

Parameter	Explanation	Value
colsample_bytree	It is the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.	0.9
eta	Step size shrinkage used in update to prevents overfitting.	0.05
gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree.	0
reg_alpha	L1 regularization term on weights. Increasing this value will make model more conservative.	0
reg_lambda	L2 regularization term on weights. Increasing this value will make model more conservative.	0
min_child_weight	Minimum sum of instance weight (hessian) needed in a child.	2

Parameter	Explanation	Value
num_parallel_tree	Number of parallel trees constructed during each iteration.	3000
max_depth	Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit.	6

6.3 LightGBM Calibration

Model Structure:

During the parameter tuning process for the LightGBM model, we concentrated on eight key parameters to derive an optimal model.

Table 8 LightGBM Parameters

Parameter	Explanation	Value
colsample_bytree	LightGBM will randomly select a subset of features on each iteration (tree) if colsample_bytree is smaller than 1.0.	0.8
learning_rate	The shrinkage rate.	0.05
num_leaves	Max number of leaves in one tree.	30
reg_alpha	L1 regularization.	0.08
reg_lambda	L2 regularization.	0.5
min_child_weight	Minimal sum hessian in one leaf.	2
sub_sample	Subsample ratio of the training instance.	0.6
max_depth	Limit the max depth for tree model.	8
n_estimators	Number of boosted trees to fit.	6000

6.4 Artificial Neural Network Calibraion

The neural network (NN) model we use is a feed-forward MultiLayer Perceptron that employs backpropagation for training. A leaky rectified linear unit (leaky ReLu) activation function is utilized, and the input layer comprises 9 nodes representing market and Heston model factors. To customize the model's performance, hyperparameters for the hidden layers were adjusted, including custom learning rates for different epochs. The final network structure consists of 1 normalization Layer and 6 hidden layers each with hidden node 64(We consider a linear layer and an activation layer as one layer), has an initial learning rate of 0.001, operates on a dataset size of 100,000, and is trained for 180 epochs over a period of 20 minutes with an early stopping callback (if the test-set losses does not improve for consecutive 5 epoch we will stop training) activated.

Table 9 ANN Model Configuration

Hyperparameters	**config**
$DatasetSize$	100,00
$BatchSize$	128
$InitialLearningRate$	0.001
$ELRA$	0.98
$HiddenLayers(Depth)$	5
$EpochSize$	180
$TrainingTime$	125ms

6.5 Model Comparison

Final Results:

		v_0	θ	κ	σ	ρ	MAE
XGBoost	European	70.2%	292.5%	87.9%	35.8%	198.9%	0.015
	American	114.7%	95.3%	88.2%	68.5%	139.8%	0.008
LightGBM	European	34.8%	135.4%	94.5%	46.7%	204.4%	0.011
	American	40.7%	27.0%	86.9%	66.1%	255.6%	0.009
ANN	European	61.9%	67.1%	39.2%	55.1%	248.9%	0.005
	American	38.5%	524.9%	79.9%	71.6%	122.3%	0.004

From the aspect of performance ANN is better than XGBoost which is better than LightGBM.In the model training results, American options perform slightly better than European options. Overall, The results of the estimation of these Heston parameters are not good enough.

7. Conclusion

In the first part of this project, we introduce several machine learning model including Random Forest, XGBoost, Cat Boost, Light GBM and ANN to generate option price. In doing so, we have outlined our data generation process, the corresponding introductions of the different machine learning models and compare their capability with respect to MAE. The result on training set shows that The ANN model displays a superior predictive power for option pricing when compared to other widely-used tree models. In the next part of the project, we try to calibrate the Heston model used to price the option price in the previous section , in this part we also use ANN, Light GBM and XGBoost for the pricers of Heston parameters and compare the MAE of different models. While we find that the calibrated heston parameters deviate greatly from the initial value, so machine learning pricers may not good enough for the pricing. For further studies, we will try to find the better model/feature for the calibration and test its performance using the real-world market data.