Pricing American Style Stock Options Using Machine Learning

by

Noah Kriss

A thesis submitted to the

Department of Physics, Engineering Physics and Astronomy

in conformity with the requirements for

the degree of Bachelor of Applied Science

Queen's University

Kingston, Ontario, Canada

April 2024

Abstract

Options contracts, particularly the more flexible American-style stock options, are very dynamic securities that support a wide range of strategic financial tactics. These options are also known for their competitive pricing requirements due to their constant fluctuating price and lack of proven pricing method. Current methods for pricing American-style stock options are computationally expensive and are only viable for large financial companies. This thesis proposes a new pricing method; a Transformer based neural network, to improve on the current pricing methods. The thesis proves that the model is much more computationally efficient than current models, and that the Transformer improves the models ability able to make correlations from the input vectors to price predictions. Although the model proved to not be accurate enough for industry use, the thesis details improvements that could be made to push the model to compete or even excel against current market strategies.

Acknowledgments

Special thanks to Professor Ryan Martin, Alan and Naomi Kriss, and Virgil Mactavish

Contents

List of Tables

List of Figures

Chapter 1

1.1 Introduction

Options trading provides unique opportunities in the finance industry to speculate on market movements, and can give unique strategic opportunities to a portfolio. Among the many different types of options, American-style options stand out due to their flexibility, being executable at any point up to their expiration date. This flexibility, gives traders further opportunities for hedging, speculation, and arbitrage. While there are many proven pricing methods for European-style stock options such as the Black-Scholes Method, there are no clearly proven pricing methods for the more flexible American-style options. Current methods such as binomial tree, and Monte-Carlo simulations are computationally intensive and can only be used by large companies with access to massive computational power. This thesis proposes using machine learning in an attempt to create a more efficient pricing method to compete with these larger financial players. The proposed method is to use a Transformer based neural network architecture to find underlying nuances in the options contract behavior.

Chapter 2

2.1 Introduction and Motivation

Pricing American-style stock options poses a unique problem due to their early exercise feature. While European-style stock options, only exercisable on the expiry date, can be priced using proven methods such as Black-Scholes, American-style options need much more computationally expensive numerical or simulation based methods. The ability to value these options accurately and quickly is critical to making informed financial decisions and preventing arbitrage loss. Stock options, particularly American-style are a very dynamic security, being used in a multitude of financial strategies. These strategies are so widely used that the options market has recently surpassed the size of the market for the underlying stocks [1]. This motivates a strong need for an accurate and reliable pricing method.

Currently, the methods used to price American-style options, such as Monte-Carlo Simulations and Binomial Trees, are only viable for large financial players because of their dependence on large computational power. This gives these players a great advantage when dealing with these securities and allows them to make large annual gains. Other securities including European-style options, stocks, and bonds, all have standard, easily executable pricing algorithms but there are no such methods for American-style options.

Recent advances in modern computing have provided a unique opportunity to address this challenge. Machine learning has become a powerful tool for addressing challenges formerly considered intractable in data science, being able to learn from vast amounts of data and identify patterns that traditional models could not capture. In 2018 Google researchers published a paper on a new architecture called the Transformer that revolutionizing data science tasks such as natural language processing, and facilitated the creation of powerful AI's such as ChatGPT and BERT. The Transformer has also been showing success in time sequential pattern recognition. This thesis proposes a Transformer based neural network architecture as an attempt to create an efficient pricing method to compete with these larger financial players.

2.2 Background

There are two topics of prerequisite information essential to understanding the content of this thesis. The first is understanding financial concepts and terminology, including basics on options and financial markets. The second topic is on machine learning and the strategies and algorithms used in the model. The following sections will cover each area of prerequisite knowledge necessary to understand the paper.

2.2.1 Financial Concepts

Options

First, it is most important to understand what an option is. An option is a contract that gives the holder the right, but not the obligation, to buy or sell an underlying asset at a specified price on or up until and a specified date. The freedom to make these trades has a value, often called the premium; this is what makes up the price of an options contract. If the contract allows the holder to make purchases on the underlying asset at the specified price it is called a "call option". If the holder is allowed to sell the underlying asset for the specified price it is called a "put option". European style stock options only allow the holder to make their trades at the expiry date of the contract, whereas American style stock options allow the holder to make trades at any time up until the expiry date. This flexibility allows holders to build further strategy into their portfolios, however with this added flexibility come a difficulty in price evaluation.

Important Options Parameters

There are a few parameters that are closely linked to the value of an option. These are the inputs to the Black-Scholes function, the standard method for pricing European-style stock options.

Strike Price

The strike price is the price that the underlying stock can be bought or sold at. If the strike price of the contract is less than the price of the stock, this is considered "in the money" while strike prices above the trading price of the stock are called "out of the money”. The lower the strike price, the more expensive a call option will be because it allows the holder to buy the underlying stock for less, vice versa for put options.

Interest Rate

Another key metric is the interest rate. The interest rate is the rate money can be borrowed at and the price of not investing money. This concept, known as the “time value of money”, stems from several factors including inflation, investment opportunities, risk and uncertainty, and preference for consumption. The interest rate is a proportional estimate of these factors, money in the future is discounted back to the present value at the interest rate. A trader has the ability to carry forward their money at this rate by investing in a security such as a treasury bill, one of the safest investments that return investments at the interest rate. Therefore, if the interest rate is higher, the price of a call option will also increase because the price of holding the underlying stock is higher. Again this is vice-versa for a put option because it becomes less expensive to hold the underlying stock and it is less enticing to invest in securities such as treasury bills.

Volatility

The volatility is a metric of how much a trading price varies with time. High volatility indicates that a stock will change dramatically over a short time in either direction. The volatility tends to be proportional to market sentiment. High volatility often occurs during periods of market turbulence or uncertainty, while low volatility occurs indicates more stable periods. The price of a call option increases with volatility, this is because of the asymmetric payoff structure of an option. This will be discussed further in the sections below.

Days to Expiry

As the day gets closer to the expiry date, the interest rate and volatility become less relevant and the price become much more closely proportional to the spread of the strike and underlying stock price. On the expiry date, the price of the option becomes the difference between strike and underlying price times the lot size (usually 100). A day far from the expiry date must factor in the variability of the underlying stock and the interest rate.

Asymmetric Payoff Structure

One of the most unique features of an option is its payoff structure. If someone owns a stock or a commodity, the payoff is directly proportional to the change in value of the underlying equity. However with options, buyers can profit from rising or declining prices of an underlying asset with limited downside risk. The most you can lose on an options trade is the premium paid.

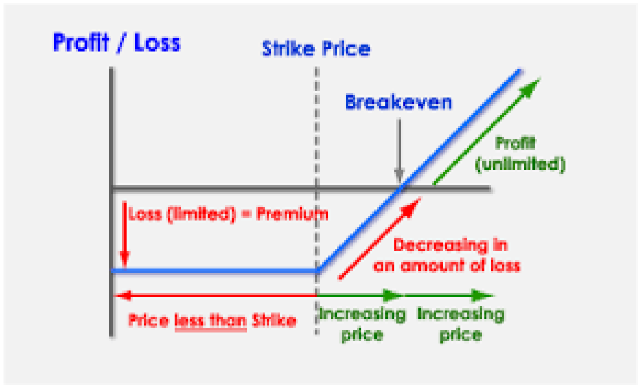


Figure 2.1: Options Contract Payoff Structure [2]

This is why the price of an option increases with volatility. Higher volatility means there will be higher swings in the underlying stock price. If the price decreases heavily, the holder of the contract will make large gains, the potential profit being unlimited. On the other hand, if the price increases greatly, the potential downside is less extreme, being limited to only the premium paid for the option contract. This payoff is the key feature that allows investors to use options for hedging against price movements or speculating while controlling potential losses.

2.2.2 Machine Learning Concepts

A main component of the network architecture used in this project is the transformer encoder, so it is important that the reader has a basic understanding of some machine learning concepts used in a transformer. A transformer is a type of neural network architecture introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017 [3]. It is the architecture that is responsible for the recent breakthroughs in deep learning power, particularly in natural language processing with models such as ChatGPT and BERT. The transformer uses a new self-attention mechanism that allows it to capture the importance of different parts of an input, and capture dependencies, contexts, and relationships of each part of the sequence. In addition, each layer in the transformer processes the input data in parallel allowing for faster and more efficient training.

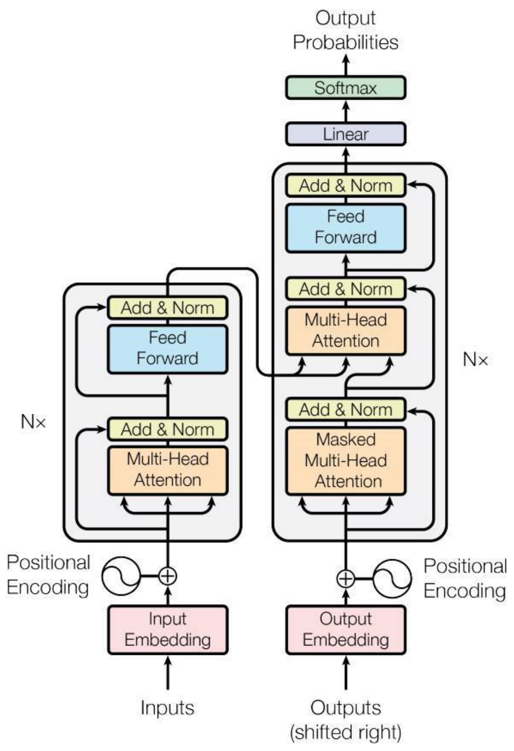


Figure 2.2: The Transformer Model Architecture [3]

This thesis seeks to use the transformer's effectiveness for sequence data processing and its computational efficiency. There are a couple layers that that are used in this thesis that would be useful to understand. The transformer introduced a new attention layer called the "multi-head attention" layer. This layer learns "attention weights" that represent the importance or relevance of different parts of the input sequence in making predictions. These weights are computed in each step of the model's training, allowing it to isolate specific elements or patterns of each set of input data. The attention mechanism learns and applies these weights multiple times in parallel, allowing the model to capture diverse patterns and relationships in the input data while maintaining training efficiency. Another layer used a lot in the project is the “dropout” layer. A dropout layer sets a random set of neurons to zero making them ignored for a given pass through. It is used to reduce the chance of overfitting the data and ensures that the model does not become too dependent on specific neurons.

2.3 Methodology

To price American-style options, this thesis proposes using a multi stage neural network based off the Transformer. The network accepts inputs of both sequential stock data and the contract data to find important parameters and underlying nuances in the data.

2.3.1 Data Preprocessing

The input data used for this project is taken from a server created by Professor Martin that scrapes market data daily, and saves the prices of options for roughly 40 tickers across a range of strike prices. It also saves other contract data including the days until expiry, current stock price, volatility, open interest, etc.

X - Sequential Data

The data from the server is processed into two parts, time sequential data and contract data. The time sequential data contains the last ten end of day stock prices and option prices arranged into a 2x10 array. A rolling window normalization technique is applied to this data, with each row being scaled individually from 0 to 1. This allows the network to find patterns and derivatives in the stock and option price behavior, while being dynamic to any range of stock prices or future market conditions.

X - Contract Data

The contract data contains the five inputs to the Black-Scholes model. These five features are added to the array and are un-normalized to preserve relative scale between vectors. This proved to be important for accuracy.

Y - Options Price

The output of the network is the predicted options price. The y-train and y-test datasets are the average of the bid and ask prices for the day. The "true price" of each contract was estimated to be the average of the bid and ask prices for the day. It was found that the model preformed very poorly at pricing options whose value was zero or near zero. Since these contracts are less interesting to price anyway, all input vectors with options prices below one dollar were removed from the training set.

Data Preparation

The above datasets were created across a set of selected Canadian tickers saved on the server. The TD options contracts were removed to be used as a separate testing dataset. The remaining tickers were split into an a training and testing split. The data set was not shuffled before training. This was important to ensure that the model was not learning the behavior of the prices on specific days and was instead learning to extract the price from the given input parameters. For example, if the model had seen a contract vector in training from the a day after a contract in validation, it could already know the price because it was part of the input parameters. This could make the model preform better in training and validation, but would make it fall short when predicting new options prices.

2.3.2 Network Architecture

The network accepts the two inputs outlined in the section above. The time sequential data is fed into a Transformer encoder. The structure of one transformer encoder is as follows. The sequential data is first passed through a normalization layer. The normalization method selected is keras's LayerNormalization, the standard for Transformer encoders. Next, the normalized data is fed through a multi-head attention layer to learn dependencies and discover patterns. The next layer is a dropout layer to reduce chances of overfitting. The output of this section is added back into the original inputs and is fed into a feedforward section. The feedforward section consists of a normal convolutional layer, a dropout layer, then another convolutional layer, then another dropout layer. This feedforward section learns more about the patterns and combines them for the final encoding. The above architecture is one transformer encoder, two of these are layered on top of each other to finish the sequential data processing. The code for an encoder can be found in the appendix.

The contract data is processed by a dense, feedforward network. The isolated processing of this data is only one layer with 20 nodes. This is sufficient because the processing of this data should only need to be as computationally complex as the Black-Scholes model.

The output of each of these sections contain 20 nodes. It was assumed that the importance of the contract data was more important than the sequential data so to assist the model in understanding this importance a dense layer was inserted on the

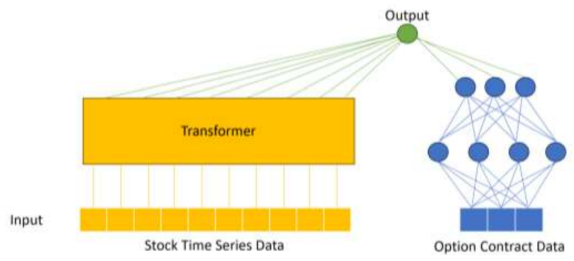


Figure 2.3: Thesis Neural Network Architecture. Note: node numbers and feature lengths are not to scale.

output of the sequential data to reduce its nodes while maintaining important information. After some testing and iteration, the best balance was found to be half the nodes of the contract data, 10 nodes. These nodes are flattened and concatenated to the 20 nodes from the contract data. The data is then run through another dense feed forward network to produce a final output. A rough diagram of the model architecture can be seen below.

2.4 Iteration

2.4.1 Normalization

Several types of normalization techniques were considered and experimented with. This experimentation occurred independently for the sequential and the contract datasets.

Sequential Dataset Normalization

Originally, the sequential dataset was scaled using keras's LayerNormalization function. This function scales then data using a formula that makes the mean of the data close to zero and the standard deviation close to one. The first issue with this normalization technique was that the scaling method was non-linear. This caused a disproportionate scaling to stock prices at the extremes, there was a loss of linearity for each input vector and it made it difficult for the model to understand relative changes between input vectors. Another option was to scale the data from 0 to 1. This better preserved the linearity, however issues remained. There is a large range of stock prices in the training set and there is little deviation in each input vector of stocks because the stock price does not usually change much over the course of 10 days. Using this normalization method meant that each input vector of sequential stock data contained an array of very compressed, similar values making it more difficult for the model to find patterns in the price behavior. Also, by scaling from 0 to 1, this assumes that the maximum/minimum input stock price the model will see is the maximum/minimum price in the training set, however this is almost certainly not the case. Stock prices vary all the time for example with companies opting to spilt shares and increasing dividends, or long term increasing trends and inflation. An option to account for these fluctuating market trends is to add a buffer for example scaling the data from -0.1 to 1.1. This ensures that values slightly outside the original range are accommodated within the scaled 0 to 1 interval without being clipped.

The best normalization technique found was a 0 to 1 rolling window normalization. Rolling window normalization scales the data across each input vector instead of the entire dataset. This works very well for this use case because the important information for the sequential data is not the actual value but the relative changes in price across the time range. This normalization technique does not compress the data, can be scaled linearly, and is future proof to any range of stock prices. Further, this technique inherently incorporates the concept that splitting shares has no effect on the value of options contracts.

Contract Data Normalization

The method for determining the best contract data normalization technique was mostly iterative. The only qualitative assumption that could be made was that a rolling window technique can not work because in this dataset it is important to preserve the true scale of the input features. After trying each normalization technique described above, doing nothing, leaving the data un-normalized proved to be most accurate. This is likely because of the importance of preserving scale between input vectors.

2.4.2 Shuffling Dataset

During preprocessing there is an option to shuffle the training set. Generally it is common practice to shuffle the dataset for a couple reasons. First, without shuffling, the neural network might learn patterns that are specific to the order of the data rather than the underlying relationships between inputs and outputs. Also, shuffling can often improve generalization. By shuffling the dataset, the model is able to train off a more diverse range of the training set so it can make better generalizations from the training data to new unseen data.

Despite the many advantages of shuffling it is common in time sequential and stock modeling to not shuffle the input data. Maintaining the order of time sequential data is crucial for making accurate predictions. Shuffling disrupts the temporal sequence "leading to a loss of the time-dependent context that the model needs to learn." Additionally, while not shuffling opens risks of the model becoming biased to tickers, shuffling the data opens risk that the model becomes biased to temporal periods. This could make the model appear better during training and against validation data because it has seen the behavior of options prices during these market conditions. However when applying the model to future data the accuracy would fall off because it has not seen data in this temporal period. Instead the model should learn to find the underlying nuances independent of past market conditions.

2.4.3 Learning Rate

A range of learning rates were tried. The learning rate is the amount the weights of the network are updated during training. A larger learning rate allows for a faster convergence and allows the model to not get stuck at local minima. A lower learning rate allows for a more stable convergence and more precise final weights.

Generally, it is common to use a learning rate of around 1e-3 for dense networks and a learning rate of around 1e-6 for Transformers. The final neural network architecture contained both a Transformer and dense feed forward sections therefore some thought went into this parameter. Upon testing it was found that a higher learning rate greatly improved training time while a lower learning rate gave a much smoother convergence. Since a transformer is a computationally expensive network element, training time is already very long.

To try to improve training time while capitalizing on the smoother convergence, learning rate scheduling was used. Learning rate scheduling is a strategy that allows the network to vary its the learning rate across training epochs. By starting the learning rate at 1e-3 and gradually reducing it to 1e-6, the model is able to train faster and reduce chances of getting stuck in local minima while increasing convergence

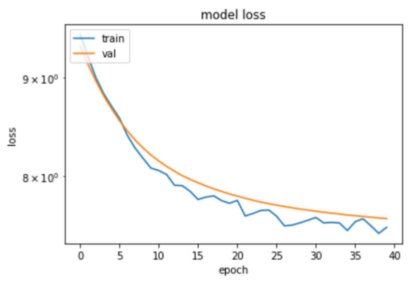


Figure 2.4: Rough Loss Curve Example

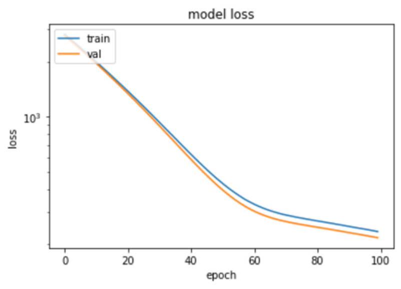


Figure 2.5: Smooth Loss Curve Example

smoothness and not overshooting minimums.

2.4.4 Transformer Encoders

The network was trained and evaluated for performance with 0 to 4 transformer encoders. It was found that with a smaller dataset for example one ticker, around 10,000 input vectors, the Transformer did not improve accuracy. Adding more Transformer encoders only increased training time. As the size of the dataset was increased, the Transformer encoders had more to train from, and networks trained with Transformers preformed better. The final training set used around 100,000 input vectors and preformed best with two encoders, however if the model were scaled it could probably use more encoders to improve accuracy. With the final dataset, adding the two encoders improved accuracy compared to removing encoders entirely.

2.4.5 Contract Data Network

Two parameters were varied to find the best shape of the contract data network, depth and number of nodes. It turns out that it does not require a very complicated network to find the patterns in the contract data, a deeper network only served to increase training time and even still was not as accurate. A single layer was sufficient.

When varying the number of nodes, the number did not seem to matter much, instead the relative output of the contract network to the sequential network mattered more. A 20 contract nodes to 10 sequential output nodes was found to be most accurate, while training efficiently.

2.5 Results

The final model was trained on the Canadian tickers on the server (except TD). The final network parameters for training can be seen in the table below.

Table 2.1: Summary of Model Parameters

Parameter	Value
Transformer Head Size	256
Number of Transformer Heads	4
Convolutional Layer Dimension	4
Multilayer Perceptron Units	40
Transformer Encoders	2
Dropout	0.1
Epochs	Max = 500
Training Set Size	89850

Training took a total of 108 epochs before early stopping took effect. Each epoch averaged around 24 seconds making the total training time 41 minutes long running on a standard CPU. This time could be greatly improved running on a GPU. The loss curves descend at a steady rate with the loss decreasing rapidly at first and leveling out by the end of training. The loss curves are fairly rough until the scheduling

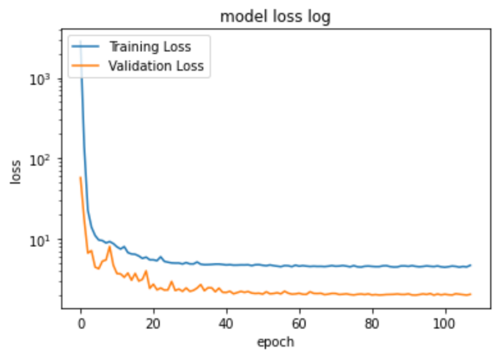


Figure 2.6: Final Training and Validation Loss Curves

reduces the learning rate and the curves smooth out.

Testing was first preformed on an array containing the Canadian tickers used in training. The model had never seen these input vectors however it had seen these tickers before during training. Several metrics were used to evaluate the final results, each revealing important information about the model success and viability. The three main metrics used are mean absolute error (MAE), mean absolute percent error (MAPE), and root mean squared error (RMSE). The formulae for these can be found below.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.1}$$

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{2.2}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.3}$$

The following table contains the model's score in each of these metrics. The mean and max absolute and percent errors are included, as well as the percent of the model's predictions that were within 10% of true.

Table 2.2: Summary of Model Performance Metrics.

Metric	Value
Mean Absolute Error	0.4232
Median Absolute Error	0.2757
Max Absolute Error	17.8682
Mean Percent Error	9.75%
Median Percent Error	4.57%
Max Percent Error	389.70%
Below 10% Error	72.98%
RMSE	0.7372

A correlation plot is also produced to better understand what the model is producing. The blue line shows the perfect line of predictions.

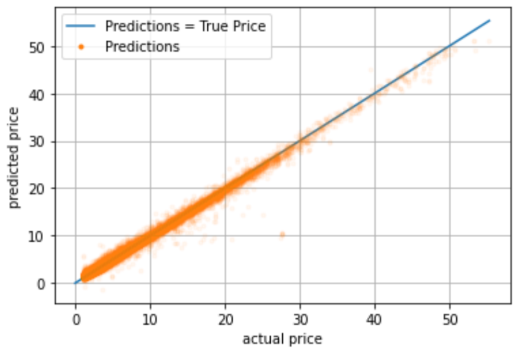


Figure 2.7: Correlation plot of model predictions vs actual option contract price

To unpack the models performance in relation to its input parameters, a plot was produced showing the percent error with respect to each feature. Each point is a prediction color coded by ticker. A sixth plot is produced that shows the error for each prediction sorted by each ticker.

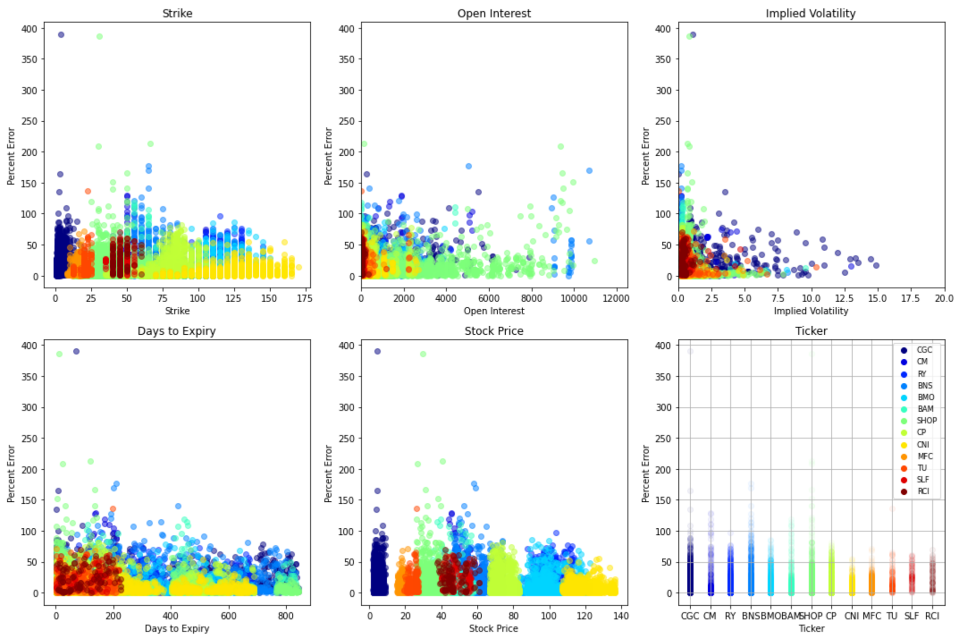


Figure 2.8: Parameter Percent Error Analysis

A second set of testing was done on the isolated TD ticker. This testing was done to evaluate how the model preforms on a ticker it has never seen before. This should be a good metric of the generality of the model; whether it can be used on any option contract for any ticker on the market, despite only being trained on a smaller set. The following table and correlation plot were produced.

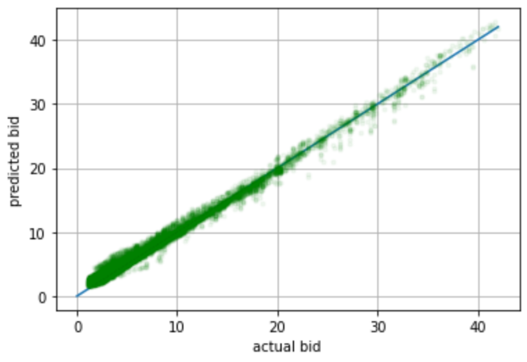


Figure 2.9: Correlation plot of model predictions on TD test set vs actual option contract price

Table 2.3: Summary of Model Performance Metrics on TD Test Set

Metric	Value
Mean Percent Error	17.49%
Median Percent Error	6.75
Max Percent Error	168.67%
Max Error (RMSE)	17.14
RMSE	0.8546
Max Error	4.5652
Percentage below 10% Error	58.95%

2.6 Discussion

2.6.1 Model Training

The loss curves indicate that the model is learning well. Both loss curves descend steadily getting smoother as the learning rate scheduling takes effect. An interesting observation can be made about the loss plot; the validation loss starts and remains below the training loss. Training loss is usually less than validation loss because the model is learning from and updating weights based off the training data. It is odd that the model could preform better on the validation data it has never seen before. This can often happen for a couple reasons the most likely in this case being that the model is too complicated for the training set. This is the first indicator that more data is needed to make this model preform to the best of its capabilities. Figure 2.7 confirms that the model is learning to correlate the input vectors to the options prices. The predictions, shown as the orange dots, closely fit the actual price line.

2.6.2 Industry Viability

When trading derivatives markets it is crucial to be confident in the valuation of securities before trades because even small deviations can result in massive losses. At the beginning of this thesis, a quantitative metric was suggested to evaluate if the model had met market standards; predictions between the bid and ask. According to the efficient market theorem, this would put the model within the ballpark of sufficiency for use in industry. The model unfortunately does not meet the standards necessary to be used in a financial application. Evaluated against this metric the model predicts within the bid ask price 23.7% of the time. However, the more important goal of the thesis is to seek a new method of pricing American-style stock options. The remainder of the paper will discuss how the model can be developed to not only become viable for market use but excel against current methods.

2.6.3 Evaluation Metrics

Table 2.5 gives a deeper view into how the model is preforming. The mean absolute error is \$0.42 across contracts valued at \$1 to around \$60. The median absolute error is similar to the mean at \$0.27 indicating that the absolute error is similar across the entire dataset. In financial applications relative change or scale is often more appropriate, so perhaps more important for this application is percent error. The percent error is much less linear across the dataset. The mean error is 9.75% with a median of 4.57%. The lower median indicates that the bulk of the testing error is below the average of 9.75%. As the actual price of the options gets closer to zero the model's relative error increases significantly. The max percent error is 389%. Given the size of the dataset and chances for anomalies this is a relatively low error compared to the average. Also, this error occurs when the price was close to zero so the absolute error was only a couple dollars overshot. The model is able to consistently predict the price within 10% of the true price at a rate of 70% of the time indicating that other than the anomalies, the model can make consistent correlations.

2.6.4 Feature Analysis

By analyzing the behavior of error with respect to each feature, we may be able to understand why the model is falling short. Five of the six categories appear to be independent of model accuracy. These are strike, open interest, days to expiry, last stock price, and ticker. The way we can tell they have little correlation to model accuracy is because there does not seem to be a trend in percent error when varying each parameter. There are some tickers that do not preform quite as well as others however these differences are minimal. The implied volatility does seem to have an error bias as it approaches zero.

Implied Volatility Error Analysis

The price of options tend to decrease with lower volatility because there is less of a chance the underlying stock will fluctuate, becoming in the money. If the volatility is low, the price of the option should be fairly steady because there is less uncertainty in the market. An interesting observation is that the highest errors are in contracts with low volatility, but tickers that tend to have higher volatility like Canopy Growth Corp (CGC) colored in purple. It is likely that despite the low volatility on this trading day, the market expects it to go back up and their pricing calculations are being adjusted. These instances are rare, so letting the model train off more data with more of these examples could help it learn to price these outliers more effectively.

2.6.5 TD Test Analysis

The TD test showed that the model is still able to make correlated predictions even when faced with new tickers, however the performance was lower than the already seen tickers. This indicates that the model was overfitting to parameters common to specific tickers. The most likely candidates are latest stock prices and strike prices. Given the good correlation, it is likely that a larger dataset with a wider range of stock and strike prices would allow the model to learn better without having to overfit.

2.6.6 Computational Efficiency

One of the other key metrics for the model is its computational efficiency. Although it is optimal for the model to train on large computers it took under an hour to train even on a CPU. If the model were trained on a deeper dataset it would take much longer but the computational power required to train the model is small compared to that needed for Monte-Carlo simulations and Binomial Trees. Further, the calculation time for the model to predict a given price is orders of magnitude faster than the other traditional methods. Below is a table comparing the speed to popular pricing methods.

Table 2.4: Model Efficiency Comparison Data

Pricing Method	Evaluation Speed [s]
Transformer Neural Network	0.000257
Finite Difference PDE	0.61[4]
Binomial Method (Constant Volatility)	7.3[4]
Longstaff-Schwartz Method	139.2[4]

It is clear that, should this method become competitively accurate to the other methods, it would quickly adapted due to opportunities for arbitrage.

2.6.7 Transformer Performance

One of the biggest successes of the thesis was showing that then Transformer had added benefits to the pricing accuracy. With low amounts of training data, the transformer was not able to find correlation and the network without any encoders preformed just as well as with encoders. However as the dataset size was increased, the Transformer began to find patterns in the sequential data and the performance increased from the zero encoder network. The table below is a performance comparison of the network with and without Transformer encoders.

Table 2.5: Comparison of Transformer Encoder Accuracy

Metric	2 Transformer Encoders	0 Transformer Encoders
Mean Absolute Error	0.4232	0.8457
Median Absolute Error	0.2757	0.6034
Max Absolute Error	17.8682	24.2525
Mean Percent Error	9.75%	22.95%
Median Percent Error	4.57%	9.73%
Max Percent Error	389.70%	293.95%
Below 10% Error	72.98%	50.69%
RMSE	0.7372	1.2984

Chapter 3

Conclusion

This thesis proposed a Transformer based neural network architecture as a method to price American-style stock options and compete with more computationally demanding methods only possible for large financial companies. The network proved that it could make consistent correlations within 10% of the true price at a rate of 73%. The Transformer portion of the network proved to significantly improve accuracy boosting the predictions within 10% by 22.3%. One of the model's biggest successes was that it is able to make predictions on the price over 2000 times faster than other pricing strategies.

While the model is not yet accurate enough to be used in an industry setting, the correlation and efficiency indicate that the model could be a very powerful pricing method. Once the accuracy of the model is competitive with other pricing strategies it would excel against competition due to its drastic prediction efficiency. The analysis indicates that the major next step to improve the model accuracy would be increasing the training dataset size allow the transformers to learn deeper correlations. A larger dataset would also allow for more transformer encoders, which proved to increase accuracy, and make the model more general, able to predict across broader range of sectors and market conditions.

Bibliography

[1] J. Detrixhe, “Gamma squeeze: Options trading is poised to overtake the stock market.” `https://qz.com/2092197/options-trading-is-poised-to-overtake-the-stock-market`, 2021. Accessed: 2024-04-03.

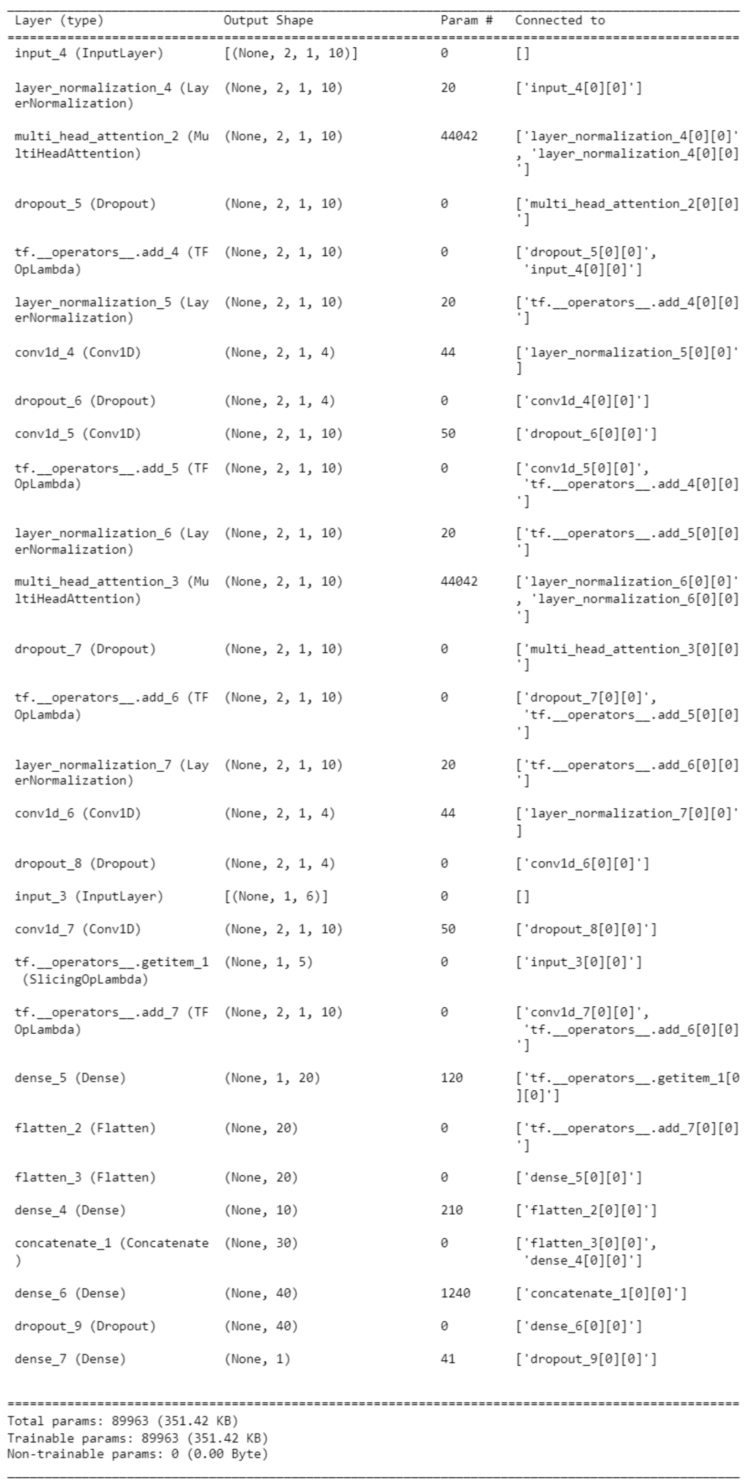
[2] Phillip Securities (Thailand) Public Company Limited, “Understanding options - phillip securities (thailand) public company limited,” 2023. [Online; accessed April-2-2024].

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *2017 Neural Information Processing Systems (NIPS)*, 2017.

[4] D. R. Anderson and U. Ulrych, “Accelerated american option pricing with deep neural networks,” *Quantitative Finance and Economics*, vol. 7, no. 2, pp. 207–228, 2023.

Appendix A

Statement of Work and Contributions



```
____________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
============================================================================================
 input_4 (InputLayer)        [(None, 2, 1, 10)]           0         []

 layer_normalization_4 (Lay  (None, 2, 1, 10)             20        ['input_4[0][0]']
 erNormalization)

 multi_head_attention_2 (Mu  (None, 2, 1, 10)             44042     ['layer_normalization_4[0][0]'
 ltiHeadAttention)                                                  , 'layer_normalization_4[0][0]
                                                                    ']

 dropout_5 (Dropout)         (None, 2, 1, 10)             0         ['multi_head_attention_2[0][0]
                                                                    ']

 tf.__operators__.add_4 (TF  (None, 2, 1, 10)             0         ['dropout_5[0][0]',
 OpLambda)                                                           'input_4[0][0]']

 layer_normalization_5 (Lay  (None, 2, 1, 10)             20        ['tf.__operators__.add_4[0][0]
 erNormalization)                                                   ']

 conv1d_4 (Conv1D)           (None, 2, 1, 4)              44        ['layer_normalization_5[0][0]'
                                                                    ]

 dropout_6 (Dropout)         (None, 2, 1, 4)              0         ['conv1d_4[0][0]']

 conv1d_5 (Conv1D)           (None, 2, 1, 10)             50        ['dropout_6[0][0]']

 tf.__operators__.add_5 (TF  (None, 2, 1, 10)             0         ['conv1d_5[0][0]',
 OpLambda)                                                           'tf.__operators__.add_4[0][0]
                                                                    ']

 layer_normalization_6 (Lay  (None, 2, 1, 10)             20        ['tf.__operators__.add_5[0][0]
 erNormalization)                                                   ']

 multi_head_attention_3 (Mu  (None, 2, 1, 10)             44042     ['layer_normalization_6[0][0]'
 ltiHeadAttention)                                                  , 'layer_normalization_6[0][0]
                                                                    ']

 dropout_7 (Dropout)         (None, 2, 1, 10)             0         ['multi_head_attention_3[0][0]
                                                                    ']

 tf.__operators__.add_6 (TF  (None, 2, 1, 10)             0         ['dropout_7[0][0]',
 OpLambda)                                                           'tf.__operators__.add_5[0][0]
                                                                    ']

 layer_normalization_7 (Lay  (None, 2, 1, 10)             20        ['tf.__operators__.add_6[0][0]
 erNormalization)                                                   ']

 conv1d_6 (Conv1D)           (None, 2, 1, 4)              44        ['layer_normalization_7[0][0]'
                                                                    ]

 dropout_8 (Dropout)         (None, 2, 1, 4)              0         ['conv1d_6[0][0]']

 input_3 (InputLayer)        [(None, 1, 6)]               0         []

 conv1d_7 (Conv1D)           (None, 2, 1, 10)             50        ['dropout_8[0][0]']

 tf.__operators__.getitem_1  (None, 1, 5)                 0         ['input_3[0][0]']
  (SlicingOpLambda)

 tf.__operators__.add_7 (TF  (None, 2, 1, 10)             0         ['conv1d_7[0][0]',
 OpLambda)                                                           'tf.__operators__.add_6[0][0]
                                                                    ']

 dense_5 (Dense)             (None, 1, 20)                120       ['tf.__operators__.getitem_1[0
                                                                    ][0]']

 flatten_2 (Flatten)         (None, 20)                   0         ['tf.__operators__.add_7[0][0]
                                                                    ']

 flatten_3 (Flatten)         (None, 20)                   0         ['dense_5[0][0]']

 dense_4 (Dense)             (None, 10)                   210       ['flatten_2[0][0]']

 concatenate_1 (Concatenate  (None, 30)                   0         ['flatten_3[0][0]',
 )                                                                   'dense_4[0][0]']

 dense_6 (Dense)             (None, 40)                   1240      ['concatenate_1[0][0]']

 dropout_9 (Dropout)         (None, 40)                   0         ['dense_6[0][0]']

 dense_7 (Dense)             (None, 1)                    41        ['dropout_9[0][0]']

============================================================================================
Total params: 89963 (351.42 KB)
Trainable params: 89963 (351.42 KB)
Non-trainable params: 0 (0.00 Byte)
____________________________________________________________________________________________
```

Figure A.1: Model Summary

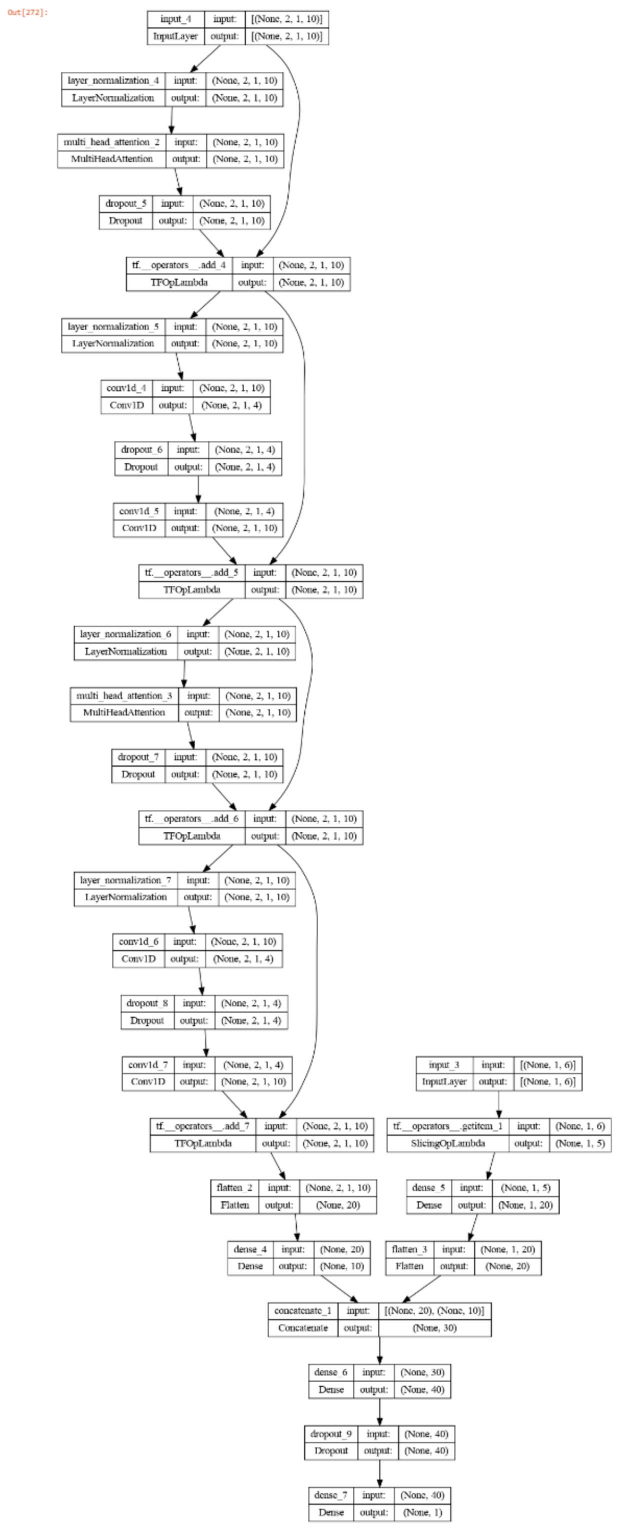
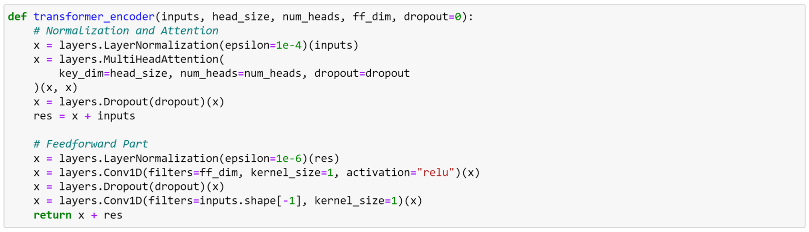

Out[272]:
input_4 | input: | [(None, 2, 1, 10)]
InputLayer | output: | [(None, 2, 1, 10)]
layer_normalization_4 | input: | (None, 2, 1, 10)
LayerNormalization | output: | (None, 2, 1, 10)
multi_head_attention_2 | input: | (None, 2, 1, 10)
MultiHeadAttention | output: | (None, 2, 1, 10)
dropout_5 | input: | (None, 2, 1, 10)
Dropout | output: | (None, 2, 1, 10)
tf.__operators__.add_4 | input: | (None, 2, 1, 10)
TFOpLambda | output: | (None, 2, 1, 10)
layer_normalization_5 | input: | (None, 2, 1, 10)
LayerNormalization | output: | (None, 2, 1, 10)
conv1d_4 | input: | (None, 2, 1, 10)
Conv1D | output: | (None, 2, 1, 4)
dropout_6 | input: | (None, 2, 1, 4)
Dropout | output: | (None, 2, 1, 4)
conv1d_5 | input: | (None, 2, 1, 4)
Conv1D | output: | (None, 2, 1, 10)
tf.__operators__.add_5 | input: | (None, 2, 1, 10)
TFOpLambda | output: | (None, 2, 1, 10)
layer_normalization_6 | input: | (None, 2, 1, 10)
LayerNormalization | output: | (None, 2, 1, 10)
multi_head_attention_3 | input: | (None, 2, 1, 10)
MultiHeadAttention | output: | (None, 2, 1, 10)
dropout_7 | input: | (None, 2, 1, 10)
Dropout | output: | (None, 2, 1, 10)
tf.__operators__.add_6 | input: | (None, 2, 1, 10)
TFOpLambda | output: | (None, 2, 1, 10)
layer_normalization_7 | input: | (None, 2, 1, 10)
LayerNormalization | output: | (None, 2, 1, 10)
conv1d_6 | input: | (None, 2, 1, 10)
Conv1D | output: | (None, 2, 1, 4)
dropout_8 | input: | (None, 2, 1, 4)
Dropout | output: | (None, 2, 1, 4)
conv1d_7 | input: | (None, 2, 1, 4)
Conv1D | output: | (None, 2, 1, 10)
input_3 | input: | [(None, 1, 6)]
InputLayer | output: | [(None, 1, 6)]
tf.__operators__.add_7 | input: | (None, 2, 1, 10)
TFOpLambda | output: | (None, 2, 1, 10)
tf.__operators__.getitem_1 | input: | (None, 1, 6)
SlicingOpLambda | output: | (None, 1, 5)
flatten_2 | input: | (None, 2, 1, 10)
Flatten | output: | (None, 20)
dense_5 | input: | (None, 1, 5)
Dense | output: | (None, 1, 20)
dense_4 | input: | (None, 20)
Dense | output: | (None, 10)
flatten_3 | input: | (None, 1, 20)
Flatten | output: | (None, 20)
concatenate_1 | input: | [(None, 20), (None, 10)]
Concatenate | output: | (None, 30)
dense_6 | input: | (None, 30)
Dense | output: | (None, 40)
dropout_9 | input: | (None, 40)
Dropout | output: | (None, 40)
dense_7 | input: | (None, 40)
Dense | output: | (None, 1)

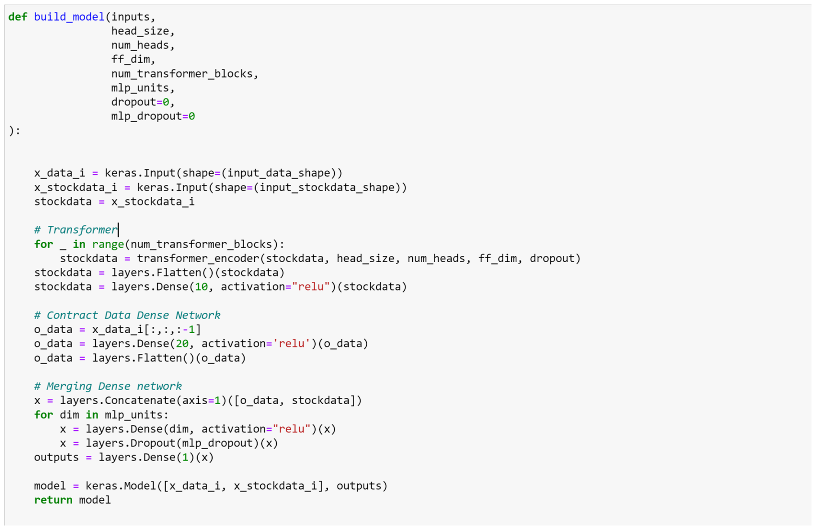
Figure A.2: Model Summary Diagram



```python
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    # Normalization and Attention
    x = layers.LayerNormalization(epsilon=1e-4)(inputs)
    x = layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout
    )(x, x)
    x = layers.Dropout(dropout)(x)
    res = x + inputs

    # Feedforward Part
    x = layers.LayerNormalization(epsilon=1e-6)(res)
    x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation="relu")(x)
    x = layers.Dropout(dropout)(x)
    x = layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
    return x + res
```

Figure A.3: Transformer Encoder Code



```python
def build_model(inputs,
                head_size,
                num_heads,
                ff_dim,
                num_transformer_blocks,
                mlp_units,
                dropout=0,
                mlp_dropout=0
):


    x_data_i = keras.Input(shape=(input_data_shape))
    x_stockdata_i = keras.Input(shape=(input_stockdata_shape))
    stockdata = x_stockdata_i

    # Transformer
    for _ in range(num_transformer_blocks):
        stockdata = transformer_encoder(stockdata, head_size, num_heads, ff_dim, dropout)
    stockdata = layers.Flatten()(stockdata)
    stockdata = layers.Dense(10, activation="relu")(stockdata)

    # Contract Data Dense Network
    o_data = x_data_i[:,:,:-1]
    o_data = layers.Dense(20, activation='relu')(o_data)
    o_data = layers.Flatten()(o_data)

    # Merging Dense network
    x = layers.Concatenate(axis=1)([o_data, stockdata])
    for dim in mlp_units:
        x = layers.Dense(dim, activation="relu")(x)
        x = layers.Dropout(mlp_dropout)(x)
    outputs = layers.Dense(1)(x)

    model = keras.Model([x_data_i, x_stockdata_i], outputs)
    return model
```

Figure A.4: Build Model Code