Neural Networks for Exotic Option Pricing

Radu Briciu
BSc Finance, PgDip Quantitative Finance

Abstract

In this paper we will explore a proposed data generation method which will construct the feature matrix for a multi-layer perceptron model designed to estimate the pricing functional of path-dependent financial derivatives. We will explore the theoretical framework and specifications for barrier and Asian option feature generation along with their respective multi-layer perceptron architectures and model performance metrics. We will demonstrate how the proposed models can retain pricing errors below one percent while reducing the computation time by up to 99.8%.

Contents

1 Introduction

In this paper we will explore a proposed method of pricing path-dependent index options via multi-layer perceptron approximations derived from the simulation of a multidimensional space representing a contract's price as a functional form of its features. Index options can generally be defined as financial derivative contracts facilitating a contingent claim on the value of a stock market index which is designed to aggregate performance across a sector or economy. It is therefore a complicated matter to evaluate the price of path-dependent option counterparts which introduce further non-linearities in their pricing functions. Path-dependent options often require statistical simulation to obtain the most accurate price, resulting in long computation times. We propose a method to generate possible values of a function which will proceed the optimization of a multi-layer perceptron model, effectively proxying the stochastic non-linear functional form between the output and it's input features. To generate a representative sample space, we calibrate historical Heston [9] parameters using market observed risk-free and dividend rates accompanied by live options trade data, thereby effectively simulating, in the case of this paper, the SPX index options market. The use of Heston's [9] stochastic volatility model as a pricing function for the underlying index allows for discreet monitoring of volatility and pricing of various market scenarios and contract types. This paper serves as a framework and demonstration of a generalized estimation process for barrier and Asian options along with a model specifications and retraining analyses. We will explore the estimation of Barrier and Asian options priced via finite difference and Monte Carlo simulation, respectively.

2 Pricing Model

2.1 Specification

To model the logarithmic price of our underlying security we use the Heston [9] model, described by the pair of stochastic differential equations:

$$dS_t = \left(r - \frac{v_t}{2}\right) dt + \sqrt{v_t}\left(\rho dW_t + \sqrt{1-\rho^2} dB_t\right) \tag{1}$$

$$dv_t = \kappa(\theta - v_t)dt + \eta\sqrt{v_t} dW_t \tag{2}$$

where

1. v_0 represents the initial variance,
2. θ is the long-run variance,
3. ρ is the correlation between the log-price process and its volatility,
4. κ is the mean reversion of the variance to θ,
5. η is the volatility of the variance process, and
6. B_t, W_t are continuous random walks.

Heston [9] famously derives the above specification as an extension to Fischer Black and Myron Scholes' [2] model for pricing options while lifting the assumption of constant volatility. The addition of an auxiliary variance process replacing the otherwise assumed constant volatility parameter σ in the renowned Black-Scholes formula allows for time-dependent, discretely measurable volatility. Consequently, implementation of the Heston [9] model for governing the underlying log price is imperative to the functionality of our model as we are estimating prices of path-dependent options which may require discrete monitoring of the spot price throughout a contract's tenor.

2.2 Historical Parameter Retrieval

We aim to create a dataset of historical parameter sets in order to simulate a market with minimal to no assumptions relating to the bounds, dispersion, or any other statistical feature of the data. Synthetic sample generation is typically more popular and has been explored in detail by Liu et. al. [11] in estimating implied volatility using artificial neural networks. However, by exploiting relatively small amounts of live trades data, one is able to attempt the automated reconstruction of volatility surfaces via discretization of the data and logic applied to trade times and volumes. One such proprietary method permitted the extraction of c. 1600 individual calibrations for as many unique live underlying spot prices from 2012 to 2024, which is far beyond the requirements for an accurate model as described by our specifications. The time continuity of data is not guaranteed, however, for the purposes of our approximation we are more concerned with ensuring each calibration surface contains enough skew in the form of multiple strikes spread both above and below the corresponding underlying price. In our method, this is accounted for by disregarding all reconstructed surfaces which do not have at least two strikes on each side of the spot price with a minimum of five contracts as needed to calibrate the Heston [9] stochastic volatility model. The optimization algorithm of choice in our study is that of the Levenberg-Marquardt algorithm as described in detail by Gavin [7] and implemented via QuantLib's Heston Model Calibration Helper.

2.2.1 Plot of parameters retrieved from SPX options trade data

Figure 1: Plot of retrieved historical Heston [9] model parameters, underlying S, risk-free, and dividend rates for the SPX options market

2.3 Barrier Option Price Functional

A Barrier option is a contingent claim on a financial security, in our case the plain European option, with the additional condition in the payoff function $C^{\text{payoff}} = (S_T - K)^+$ determining whether, at any point in the contract's tenor, the underlying spot price 'Knocks' 'In' or 'Out' (i.e., whether it breaks the barrier level):

$$\begin{aligned} C^{\text{payoff}}_{\text{UpIn}} &= \mathbb{1}_{\{S_t > B \,\forall\, t\}} \times (S_T - K)^+, \\ C^{\text{payoff}}_{\text{UpOut}} &= \mathbb{1}_{\{S_t < B \,\forall\, t\}} \times (S_T - K)^+, \\ C^{\text{payoff}}_{\text{DownIn}} &= \mathbb{1}_{\{S_t < B \,\forall\, t\}} \times (S_T - K)^+, \text{ and} \\ C^{\text{payoff}}_{\text{DownOut}} &= \mathbb{1}_{\{S_t > B \,\forall\, t\}} \times (S_T - K)^+ \end{aligned} \tag{3}$$

in the case of a European call option. Assuming we have the appropriate pricing function for one contract, we will then create a vectorized version C^{Barrier} which can be defined as:

$$C^{\text{Barrier}} = F_t(S_0, \kappa, \theta, \rho, \eta, v_0, r, g, K, B, R, D^{\text{call/put}}, D^{\text{DownIn/DownOut/UpIn/UpOut}}) \tag{4}$$

Where the price of one Barrier option contract C^{Barrier} is a function of

1. the underlying security price S (1),
2. the mean reversion speed κ of the variance process to θ (2),
3. the long-run mean of the variance process θ (2),
4. the correlation ρ between the variance process v_t and the underlying price process X_t (1),
5. the volatility of the variance process η (2),
6. the initial volatility v_0 (2),
7. the risk-free rate r,
8. the dividend rate g,
9. the strike price K,
10. the barrier level B,
11. the rebate R,
12. a logical operator $D^{\text{call/put}}$ to denote the underlying European option payoff function, and
13. a logical operator $D^{\text{DownIn/DownOut/UpIn/UpOut}}$ denoting the barrier contract type (3),

Barrier options are accurately priced using Monte Carlo simulation with control variates. However, to use this as a benchmark, we will first generate less accurate prices using a finite difference advection technique to save computational effort for our initial investigations into data generation, train set creation, and model testing.

2.4 Asian Option Price Functional

The Asian option is similarly a derivative of, in our case, a plain European option with payoff $(S_T - K)^+$ for call, and $(K - S_T)^+$ for put options. A supplementary condition is applied, defining the respective payoff functions as:

$$C_t^{\text{Arithmetic}} = e^{-r(T-t)} \times \frac{1}{m} \sum_{i=1}^{m} (S_T^{\text{Arithmetic}} - K)^+ \tag{5}$$

$$C_t^{\text{Geometric}} = e^{-r(T-t)} \times \frac{1}{m} \sum_{i=1}^{m} (S_T^{\text{Geometric}} - K)^+ \tag{6}$$

where $S_T^{\text{Arithmetic}}$ and $S_T^{\text{Geometric}}$ are the arithmetic and geometric mean of the underlying spot price measured over predetermined points in time (i.e., the 'fixing' dates). It is also possible to price an Asian option which has past fixings (i.e., is not a fresh contract), However, for the purposes of this paper, we will only be considering Asian options at inception.

$$S_T^{\text{Arithmetic}} = \frac{1}{n} \sum_{i=1}^{n} S_t^i, \quad n \in \mathbb{N},\ t_n = T \tag{7}$$

$$S_T^{\text{Geometric}} = \sqrt[n]{\prod_{i=1}^{n} S_t^i}, \quad n \in \mathbb{N},\ t_n = T \tag{8}$$

Consequently, the vectorized Asian option pricing function can be defined as:

$$C^{\text{Asian}} = F_t(S_0, \kappa, \theta, \rho, \eta, v_0, r, g, K, n, P, D^{\text{call/put}}, D^{\text{Arithmetic/Geometric}}) \tag{9}$$

where the price of an Asian option is a function of:

1. the underlying security price S (1),
2. the mean reversion speed κ of the variance process to θ (2),
3. the long-run mean of the variance process θ (2),
4. the correlation ρ between the variance process v_t and the underlying price process X_t (1),
5. the volatility of the variance process η (2),
6. the initial volatility v_0 (2),
7. the risk-free rate r,
8. the dividend rate g,
9. the strike price K,
10. the number of fixing dates n,
11. the number of past fixings$P$ which for our current purposes is always equal to zero,
12. a logical operator $D^{\text{call/put}}$ to denote the underlying European option payoff function, and
13. a logical operator $D^{\text{Arithmetic/Geometric}}$ denoting the averaging type (2.4),

3 Data Generation Method

The proposed method for amplifying a matrix of variables into a higher dimensional space is to iterate through every row i of the matrix, for each one generating a subset X_t of the final feature matrix X. Suppose we consider a set of variables:

$$\mathcal{D} = \begin{bmatrix} \mathcal{D}_{1,1} & \cdots & \mathcal{D}_{1,j} & \cdots & \mathcal{D}_{1,m} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ \mathcal{D}_{t,1} & \cdots & \mathcal{D}_{t,j} & \cdots & \mathcal{D}_{t,m} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ \mathcal{D}_{n,1} & \cdots & \mathcal{D}_{n,j} & \cdots & \mathcal{D}_{n,m} \end{bmatrix} \tag{10}$$

We will amplify the initial feature matrix $\mathcal{D}$ by iterating along the row dimension, for each iteration constructing a set of vectors H_t representing the possible values of all features individually:

$$H_t = \{ \left[\mathcal{D}_{t,1}\right], \dots, \left[\mathcal{D}_{t,j}\right], \dots, \left[\mathcal{D}_{t,m}\right], F^{(1)}_{\mathcal{D}_t}, \dots F^{(j)}_{\mathcal{D}_t}, \dots, F^{(\tilde{m})}_{\mathcal{D}_t} \} \tag{11}$$

where each $F^{(j)}$ is a vector of chosen, possible auxiliary feature values which are dependent on the state of $\mathcal{D}_t$ for every observation i and can also be a function of any feature j in $\mathcal{D}_t$. This allows us to combine vectors of varying lengths and consistently obtain all combinations of their values. The feature matrix X_t can then be defined as the Cartesian product of all elements in H_t:

$$X_t = \{(H^{(1)}_t, \dots, H^{(j)}_t, \dots, H^{(k)}_t) \mid H^{(1)}_t \in H^{(1)}_t, \dots, H^{(j)}_t \in H^{(j)}_t, \dots, H^{(k)}_t \in H^{(k)}_t \} \tag{12}$$

where X_t has dimension $(k \times s)$. Naturally occurring identities of this construction are that $k = m + \tilde{m}$ and $s = \prod_{j=1}^{k} \dim(H^{(j)}_t)$. The creation of one feature matrix X_t is, in essence, a simulation of the specified system for one observation of possible and feasible feature states in t. The combination of all X_t for n observations of the system's condition at every t will result in the final feature matrix X which we will be using for the models' initial development according to the specifications in 4.1 and 4.2.

4 Application of data generation method 3 to path-dependent index options

In the spirit of Liu et. al. [11] and Frey et. al. [6] we will generate a development dataset by simulating possible parameter combinations for a given security. Liu et. al. [11] demonstrate a considerable increase in computational efficiency with retention of low errors for the estimation of implied volatilities via artificial neural networks by considering the relative spot price (i.e., the spot price S divided by the strike price K) and the relative option price (i.e., the option's price C divided by its strike K) as opposed to their levels, a method we will be borrowing for our estimation. Frey et. al. [6] propose a data generation method via Cartesian product to create a multi-dimensional sample space representing an option's price as a functional form of its features. We therefore propose the augmented data generation method 3 utilizing Cartesian product to generate path-dependent contract features while considering the option's relative price (C/K) and any other linear features also scaled by the strike price K.

It can be said in more general terms, that we will collect a matrix $\mathcal{D}$ (10) representing market conditions to be augmented with auxiliary variables $F^{(j)}_{\mathcal{D}_t}$ (11), in essence simulating all but strictly pheasible contract combinations. We augment this sample space to cover all possible feature values of interest against which we train a neural network to obtain a highly efficient numerical approximation of the pricing function used to generate our target variable.

4.1 Defining the market condition parameter matrix $\mathcal{D}$ (10)

Following the collection of n Heston [9] model parameter observations in time i, we are able to define the initial feature matrix $\mathcal{D}$ (10) as:

$$\mathcal{D}^{\text{Heston}} = \begin{bmatrix} S_1 & \kappa_1 & \theta_1 & \rho_1 & \eta_1 & v_1 & r_1 & g_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ S_t & \kappa_t & \theta_t & \rho_t & \eta_t & v_t & r_t & g_t \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ S_n & \kappa_n & \theta_n & \rho_n & \eta_n & v_n & r_n & g_n \end{bmatrix} \tag{13}$$

consisting of n historically observed sets of Heston [9] parameters as in (1) and (2), underlying spot prices S, risk-free rates r, and dividend rates g where S and v_0 are represented as S and v respectively to simplify notation. An approach drawing from Liu. et. al. [11].

4.2 Contract Specific Methods

4.2.1 Constructing X^{Barrier}

In order to simulate the multidimensional space representing a Barrier option's price as a function of its features, we begin by iterating through the row dimension of $\mathcal{D}^{\text{Heston}}$ (13), for each row constructing a set H_t (11) with auxiliary variables $F^{(j)}_{\mathcal{D}_t}$ defined as:

$$\begin{aligned} F^{(1)}_{\mathcal{D}_t} &= K_S = \left[k_1, \dots, k_j, \dots, k_h\right], \quad k_1 < S < k_h, \quad k_{\tilde{j}} < k_j \,\forall\, \tilde{j} < j, \\ F^{(2)}_{\mathcal{D}_t} &= T = \left[\tau_1, \dots, \tau_t, \dots, \tau_h\right], \\ F^{(3)}_{\mathcal{D}_t} &= B = \left[B_1, \dots, B_j, \dots, B_h\right], \quad B_1 < S < B_h, \quad B_{\tilde{j}} < B_j \,\forall\, \tilde{j} < j, \\ F^{(4)}_{\mathcal{D}_t} &= R = [0], \\ F^{(5)}_{\mathcal{D}_t} &= F^{\text{call/put}} = \left[D^{\text{call}}, D^{\text{put}}\right], \\ F^{(6)}_{\mathcal{D}_t} &= F^{\text{Out/In}} = \left[D^{\text{Out}}, D^{\text{In}}\right], \end{aligned} \tag{14}$$

where

1. K_S is a set of strikes spread around the spot,
2. T is a set of maturities,
3. B is a set of barrier levels,
4. R is a set of rebates, which for the purposes of this study is a set consisting of only the element 0 (zero),
5. $F^{\text{call/put}}$ is a set of categorical variables representing the type of underlying European option, and
6. $F^{\text{Out/In}}$ is a set of categorical variables representing the Barrier option type payoff.

It is to be noted that the only feasible combinations are 'Down' options with $B < S$ and 'Up' with $B > S$ where B is the Barrier level and S is the underlying S_t at time i. The subset feature matrix X^{Barrier}_t will subsequently be defined as:

$$X^{\text{Barrier}}_t = \begin{bmatrix} S_t & \kappa_t & \theta_t & \rho_t & \eta_t & v_t & r_t & g_t & k_1 & \tau_1 & b_1 & r_1^{\text{rebate}} & D_1^{\text{call/put}} & D_1^{\text{barrier type}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ S_t & \kappa_t & \theta_t & \rho_t & \eta_t & v_t & r_t & g_t & k_s & t_s & n_s & r_s^{\text{rebate}} & D_s^{\text{call/put}} & D_s^{\text{barrier type}} \end{bmatrix} \tag{15}$$

essentially allowing us to generate k contracts at every observation across the rows (discrete observations in time) of our initial feature matrix $\mathcal{D}^{\text{Heston}}$ (13) where the features carried over into H_t (11) from $\mathcal{D}_t$ (10) are constant.

4.2.2 Constructing X^{Asian}

In the case of Asian options, the feature matrix generation process is not as straight forward. In order to simulate an evenly distributed sample space, we need to take into account the relationship between an Asian option's maturity and its fixing dates. This will require careful construction of a maturities vector T, ensuring the congruency of feature combinations. Therefore, we must consider a set of maturities defined as

$$T^{\text{Asian}} = [\tau_1, \dots, \tau_i, \dots, \tau_h] \tag{16}$$

which will proceed the creation of $\dim(T)$ subsets within $X_t(12)$. Subsequently, every subset is generated by considering the $H_t^{(i)}$ (11) feature set with auxiliary variables $F_{\mathcal{D}_t}^{(j)}$ defined as:

$$\begin{aligned}
F_{\mathcal{D}_t}^{(1)} &= T = [\tau_i], \\
F_{\mathcal{D}_t}^{(2)} &= K_S = [k_1, \dots, k_i, \dots, k_h], \quad k_1 < S < k_t, \quad k_{\tilde{j}} < k_j \,\forall\, \tilde{j} < j, \\
F_{\mathcal{D}_t}^{(3)} &= A_\tau = [a_1, \dots, a_i, \dots, a_h], \quad a_h \leq \tau \,\forall\, i, \\
F_{\mathcal{D}_t}^{(4)} &= P = [0], \\
F_{\mathcal{D}_t}^{(5)} &= F^{\text{call/put}} = \left[D^{\text{call}}, D^{\text{put}}\right], \\
F_{\mathcal{D}_t}^{(6)} &= F^{\text{arithmetic/geometric}} = \left[D^{\text{arithmetic}}, D^{\text{geometric}}\right].
\end{aligned} \tag{17}$$

where

1. τ is the iterated maturity in T^{Asian} (16),
2. K_S is a set of strikes spread around the spot price S,
3. A_τ are factors of τ which will determine the number of fixing dates applicable to the contract,
4. P is a set of past fixings, which for the purposes of this study is a set consisting of only the element 0 (zero),
5. $F^{\text{call/put}}$ is a set of categorical variables representing the type of underlying European option, and
6. $F^{\text{arithmetic/geometric}}$ is a set of categorical variables representing the contract's averaging type.

The subset feature matrix X_t^{Asian} will subsequently be defined as:

$$X_t^{\text{Asian}} = \begin{bmatrix} S_t & \kappa_t & \theta_t & \rho_t & \eta_t & v_t & r_t & g_t & k_1 & \tau_1 & a_1 & p_1 & D_1^{\text{call/put}} & D_1^{\text{arithmetic/geometric}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ S_t & \kappa_t & \theta_t & \rho_t & \eta_t & v_t & r_t & g_t & k_s & \tau_s & a_s & p_s & D_s^{\text{call/put}} & D_s^{\text{arithmetic/geometric}} \end{bmatrix} \tag{18}$$

essentially allowing us to generate s contracts at every observation across the rows (discrete observations in time) of our initial feature matrix $\mathcal{D}^{\text{Heston}}$ (13) where the features carried over into H_t (11) from $\mathcal{D}_t$ (10) are constant.

5 Model Training

With adequate data, we may begin training our multi-layer perceptron network to numerically approximate the relationship between our feature matrix X and target vector y.

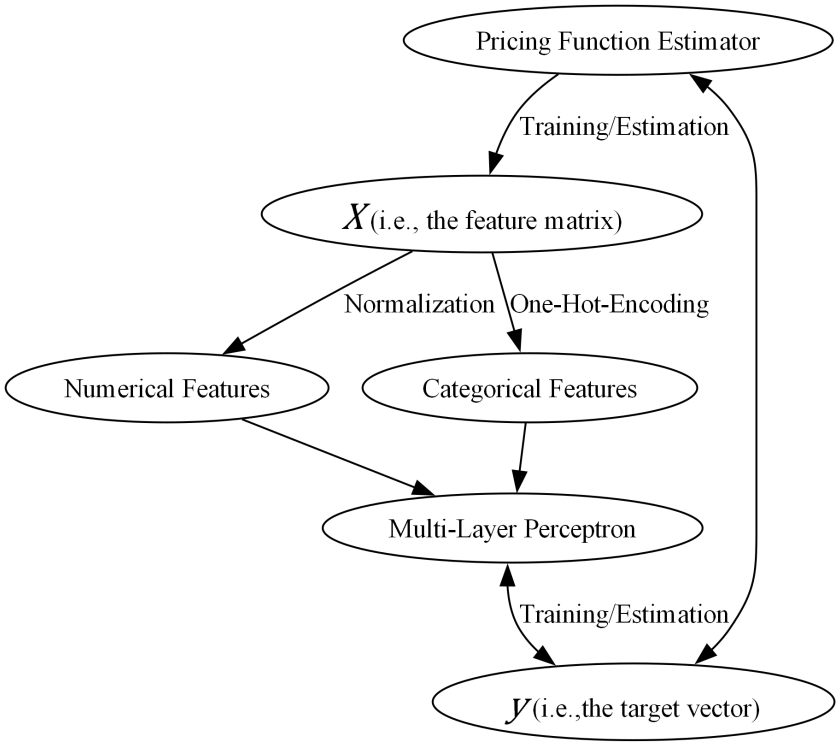
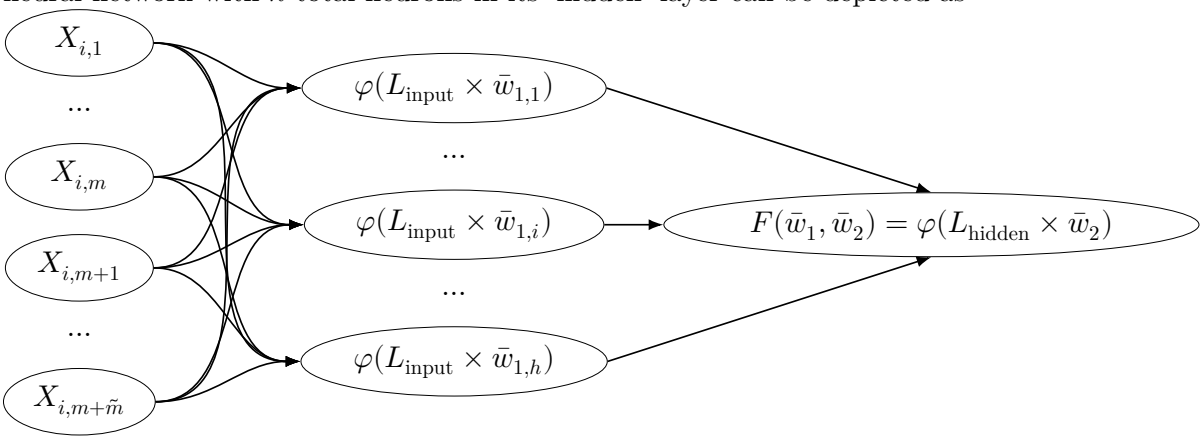


Figure 2: Graph of multi-layer perceptron Pricing Estimator

5.1 Multi-layer perceptron Specification

Technically, the neural network itself is a directed acyclic graph whose architecture and parametrization varies depending on the data being estimated. A general representation of a three-layer neural network with h total neurons in its 'hidden' layer can be depicted as



where φ is the activation function ReLU, L_{input} is the vector of normalized input features, and L_{hidden} is the vector of node values weighted by the hidden layer. Solving the model involves minimization of a loss function, in our case the squared error between the estimated $\hat{y}_i$ and the target y_i for all $N = n \times s$ observations in our feature matrix X:

$$\min_{w_1, w_2} \frac{1}{N} \sum_{i=1}^{N} \left(F_t(w_1, w_2) - y_t\right)^2 \tag{19}$$

We begin constructing the neural network as depicted in figure 2, choosing appropriate inputs and general architecture given our intentions and mathematical intuition. However, to obtain at least quasi-pertinent results, one needs to establish reasonable solving parameters.

5.2 Asian Option Network

feature	transformation
days_to_maturity	StandardScaler
fixing_frequency	StandardScaler
past_fixings	StandardScaler
risk_free_rate	StandardScaler
dividend_rate	StandardScaler
kappa	StandardScaler
theta	StandardScaler
rho	StandardScaler
eta	StandardScaler
v0	StandardScaler
relative_spot	StandardScaler
averaging_type	OneHotEncoder
w	OneHotEncoder

parameter	specification
activation	relu
solver	sgd
alpha	0.0001
batch_size	auto
learning_rate	adaptive
learning_rate_tnit	0.1
power_t	0.5
max_tter	500
loss	squared_error
hidden_layer_sizes	(20,)
shuffle	True
random_state	710
tol	0.0001
verbose	False
warm_start	False
momentum	0.9
nesterovs_momentum	True
early_stopping	True
validation_fraction	0.1
beta_1	0.9
beta_2	0.999
epsilon	1e-08
n_tter_no_change	20
max_fun	15000

5.3 Barrier Option Network

feature	transformation
days_to_maturity	StandardScaler
dividend_rate	StandardScaler
risk_free_rate	StandardScaler
theta	StandardScaler
kappa	StandardScaler
rho	StandardScaler
eta	StandardScaler
v0	StandardScaler
relative_spot	StandardScaler
relative_barrier	StandardScaler
relative_rebate	StandardScaler
barrier_type_name	OneHotEncoder
w	OneHotEncoder

parameter	specification
activation	relu
solver	sgd
alpha	0.0001
batch_size	auto
learning_rate	adaptive
learning_rate_tnit	0.1
power_t	0.5
max_tter	500
loss	squared_error
hidden_layer_sizes	(10,)
shuffle	True
random_state	710
tol	0.0001
verbose	False
warm_start	False
momentum	0.9
nesterovs_momentum	True
early_stopping	True
validation_fraction	0.1
beta_1	0.9
beta_2	0.999
epsilon	1e-08
n_tter_no_change	20
max_fun	15000

We achieve the above parametrizations after checking 6480 different parameter combinations determined by a grid of values we consider to be intuitively reasonable. While this method is severely computationally heavy, it allows both in-sample and out-of-sample errors to remain below five percent for Asian options, with in-sample error being usually around fifty basis points.

6 Model Testing

6.1 Model Performance

Figure 3: Out-of-sample average errors for barrier options

Figure 4: Out-of-sample average errors for Asian options

We notice that error increases without retraining of the model, likely due to it encountering previously unseen parameter value combinations.

6.2 Shapeliness of values

The neural network estimator can accurately produce the distribution of relative prices in the training set.

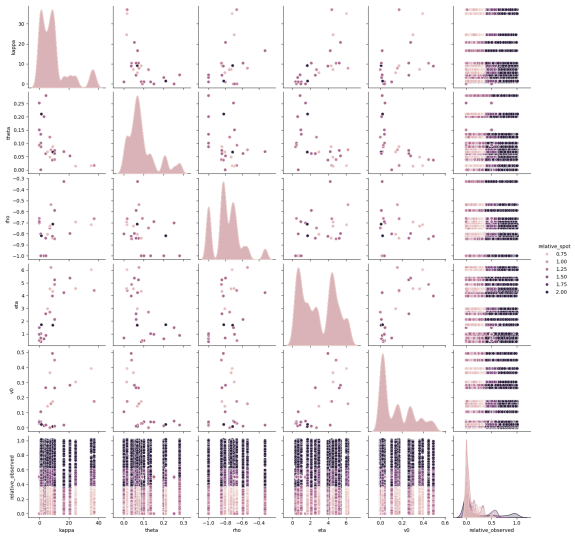


Figure 5: Joint distribution of Barrier Option prices and Heston [9] model paramters

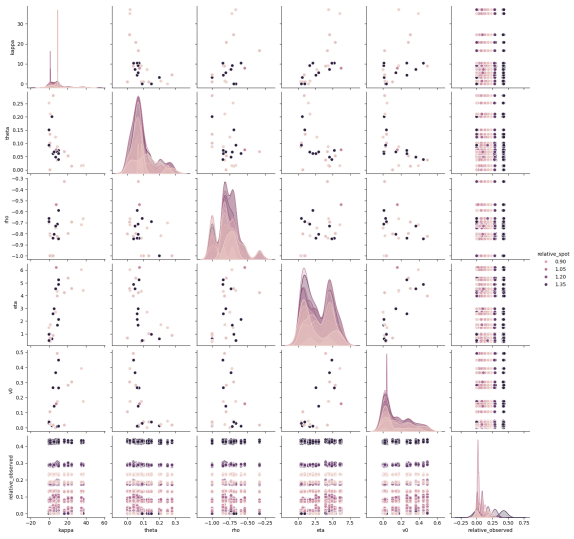


Figure 6: Joint distribution of Asian Option prices and Heston [9] model paramters

7 Concluding Remarks

In this study, we explored a data generation routine that yields parsimonious training sets for the estimation of non-linear stochastic functions. We first looked into generating training sets for barrier and Asian options priced using advection and Monte Carlo respectively.

In essence, what we have achieved is the numerical approximation of a solution to the Heston [9] pricing model using a dense neural network. The numerical approximation can have an increase in computational efficiency of up to 99.8% with in-sample errors below one percent.

8 Software Repositories

1. *Machine Learning Option Pricing.*
   https://github.com/boomelage/machine-learning-option-pricing
2. *Option Generator.*
   https://github.com/boomelage/OptionGenerator
3. *QuantLib Pricing Library.*
   https://www.quantlib.org
4. *quantlib pricers.*
   https://github.com/boomelage/quantlib_pricers
5. *sklearn convencience wrappers "convsklearn".*
   https://github.com/boomelage/convsklearn

References

[1] Christian Beck, Weinan E, and Arnulf Jentzen. "Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations". In: *Journal of Nonlinear Science* (Jan. 2019). DOI: 10.1007/s00332-018-9525-3.

[2] Fischer Black and Myron Scholes. "The pricing of options and corporate liabilities". In: *Journal of Political Economy* 81 (1973), pp. 637–654. URL: https://www.jstor.org/stable/1831029?origin=JSTOR-pdf.

[3] Robert Callan. *The essence of neural networks*. Prentice Hall Europe, 1999.

[4] Aleš Černý. *Mathematical techniques in finance: Tools for incomplete markets*. 1st ed. Princeton University Press, 2004.

[5] Aleš Černý, Christoph Czichowsky, and Jan Kallsen. "Numeraire-invariant quadratic hedging and Mean–Variance portfolio allocation". In: *Mathematics of Operations Research* 49 (Jan. 2023), pp. 752–781. DOI: 10.1287/moor.2023.1374.

[6] Christoph Frey et al. *Option pricing via machine learning with python*. 2022. URL: https://www.tidy-finance.org/python/option-pricing-via-machine-learning.html.

[7] Henri Gavin. *The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems*. 2024. URL: https://people.duke.edu/~hpgavin/lm.pdf.

[8] Paul Glasserman. *Monte carlo methods in financial engineering*. Springer Science & Business Media, Jan. 2013.

[9] Steven L Heston. "A closed-form solution for options with stochastic volatility with applications to bond and currency options". In: *Review of Financial Studies* 6 (Jan. 1993), pp. 327–343. DOI: 10.1093/rfs/6.2.327.

[10] Willem Hundsdorfer et al. "A positive finite-difference advection scheme". In: *Journal of Computational Physics* 117 (Jan. 1995), pp. 35–46. DOI: 10.1006/jcph.1995.1042.

[11] Shuaiqiang Liu, Cornelis Oosterlee, and Sander Bohte. "Pricing options and computing implied volatilities using neural networks". In: *Risks* 7 (Jan. 2019), p. 16. DOI: 10.3390/risks7010016.

[12] Terence Parr and James D Wilson. "Partial dependence through stratification". In: *Machine Learning with Applications* 6 (Jan. 2021), p. 100146. DOI: 10.1016/j.mlwa.2021.100146.

[13] Matt Pharr. *GPU gems 2 : programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley, 2004.

[14] Zixuan Zhang et al. "Bi-level optimisation of subsidy and capacity investment under competition and uncertainty". In: *European Journal of Operational Research* 318 (Jan. 2024), pp. 327–340. DOI: 10.1016/j.ejor.2024.03.028.

A Appendix

A.1 Example generation of one X_t (12) for Barrier options

```
import pandas as pd
import numpy as np
from itertools import product

s, r, g = 5785, 0.040569, 0.01283
kappa, theta, rho, eta, v0 = 3.676912, 0.255154, -1.0, 0.917628,
    0.012088
K = np.linspace(s*0.8,s*1.2,3,dtype=int)
T = [30,60]
B = np.linspace(s*0.5,s*1.2,4,dtype=int)
R = [0]
W = ['call','put']
outin = ['Out','In']

Xt = pd.DataFrame(
    product(
        [s],[r],[g],[kappa],[theta],[rho],[eta],[v0],
        K,B,R,T,W,outin
    ),
    columns=['spot_price', 'risk_free_rate', 'dividend_rate', 'kappa',
        'theta', 'rho', 'eta', 'v0', 'strike_price', 'barrier',
        'rebate', 'days_to_maturity', 'w','outin']
)
Xt['updown'] = Xt.apply(lambda row: 'Down' if row['barrier'] <
    row['spot_price'] else 'Up', axis=1)
Xt['barrier_type_name'] = Xt['updown']+Xt['outin']
Xt = Xt.drop(columns=['updown','outin'])
```

The above is an example generation of one X_t (12) in python, with H_t (11) defined as:

$$H_t^{\text{Barrier}} = \{\, \texttt{[s]}, \texttt{[r]}, \texttt{[g]}, \texttt{[kappa]}, \texttt{[theta]}, \texttt{[rho]}, \texttt{[eta]}, \texttt{[v0]}, \texttt{K}, \texttt{T}, \texttt{B}, \texttt{R}, \texttt{W}, \texttt{outin} \,\} \tag{20}$$

where X_t is stored in `Xt`, which is a pandas `DataFrame` containing the Cartesian `product` of all elements in H_t^{Barrier} (20):

	S	r	g	κ	θ	ρ	η	v	K	B	R	T	$D^{\text{call/put}}$	$D^{\text{barrier type}}$	y	cpu time
0	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	4628	2892	0	30	call	DownOut	1167.582527	0.038519
1	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	4628	2892	0	30	call	DownIn	0.639513	0.075544
2	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	4628	2892	0	30	put	DownOut	1.246778	0.036518
3	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	4628	2892	0	30	put	DownIn	0	0.071448
4	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	4628	2892	0	60	call	DownOut	1193.481261	0.038519
5	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	4628	2892	0	60	call	DownIn	0.703951	0.075544
6	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	4628	2892	0	60	put	DownOut	17.184470	0.034516
7	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	4628	2892	0	60	put	DownIn	0.662394	0.073542
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
88	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	6942	6942	0	30	call	UpOut	0	0.031117
89	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	6942	6942	0	30	call	UpIn	0	0.067470
90	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	6942	6942	0	30	put	UpOut	1139.981110	0.032102
91	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	6942	6942	0	30	put	UpIn	0.510313	0.069538
92	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	6942	6942	0	60	call	UpOut	0	0.033351
93	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	6942	6942	0	60	call	UpIn	0	0.068536
94	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	6942	6942	0	60	put	UpOut	1123.015788	0.032325
95	5785	0.040569	0.012830	3.676912	0.255154	-1	0.917628	0.012088	6942	6942	0	60	put	UpIn	0.125694	0.066511

and y (i.e., the finite difference barrier option price) is the target variable vector for its X_t (12) matrix given the feature set in H_t^{Barrier} (20). Note the $D^{\text{barrier type}}$ dummy is obtained by applying the logic `'Down' if row['barrier'] < row['spot_price'] else 'Up'`.

A.2 Example generation of one X_t (12) for Asian options

```
import pandas as pd
import numpy as np
from itertools import product

s, r, g = 5785, 0.040569, 0.01283
kappa, theta, rho, eta, v0 = 3.676912, 0.255154, -1.0, 0.917628,
    0.012088
K = np.linspace(s*0.5,s*1.5,3,dtype=int)
past_fixings = [0]
fixing_frequencies = [7,28,84]
maturities = [7,28,84]
feature_list = []
for i,t in enumerate(maturities):
    for a in fixing_frequencies[:i+1]:
        df = pd.DataFrame(
            product(
                [s],[r],[g],[kappa],[theta],[rho],[eta],[v0],
                K,[t],[a],[0],
                ['geometric','arithmetic'],['call','put']
            ),
            columns = [
                'spot_price','risk_free_rate','dividend_rate',
                'kappa','theta','rho','eta','v0',
                'strike_price','days_to_maturity',
                'fixing_frequency','past_fixings',
                'averaging_type','w'
            ]
        )
        feature_list.append(df)
Xt = pd.concat(feature_list,ignore_index=True)
```

$$H_t^{\text{Asian}} = \{\, \texttt{[s]}, \texttt{[r]}, \texttt{[g]}, \texttt{[kappa]}, \texttt{[theta]}, \texttt{[rho]}, \texttt{[eta]}, \texttt{[v0]}, \texttt{K}, \texttt{[t]}, \texttt{[a]}, \texttt{P}, \texttt{W}, \texttt{D} \,\} \tag{21}$$

	S	r	g	κ	θ	ρ	η	v	K	τ	a	P	$D^{\text{averaging type}}$	$D^{\text{call/put}}$	y	cpu
0	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	4049	7	7	0	geometric	call	1736.095472	0.150906
1	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	4049	7	7	0	geometric	put	0.000000	0.148228
2	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	4049	7	7	0	arithmetic	call	1736.289174	0.154353
3	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	4049	7	7	0	arithmetic	put	0.000000	0.154353
4	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	5785	7	7	0	geometric	call	26.706212	0.156515
5	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	5785	7	7	0	geometric	put	25.260595	0.150505
6	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	5785	7	7	0	arithmetic	call	26.777674	0.150906
7	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	5785	7	7	0	arithmetic	put	25.138355	0.147659
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
64	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	5785	84	84	0	geometric	call	151.971415	1.665390
65	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	5785	84	84	0	geometric	put	143.465505	1.666405
66	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	5785	84	84	0	arithmetic	call	155.634898	1.646350
67	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	5785	84	84	0	arithmetic	put	137.120140	1.651353
68	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	7520	84	84	0	geometric	call	0.000000	1.660147
69	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	7520	84	84	0	geometric	put	1710.370773	1.660588
70	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	7520	84	84	0	arithmetic	call	0.000000	1.658991
71	5785	0.040569	0.012830	3.676912	0.255154	-1.000000	0.917628	0.012088	7520	84	84	0	arithmetic	put	1700.361924	1.662887

A.3 Illustrative Volatility Surface

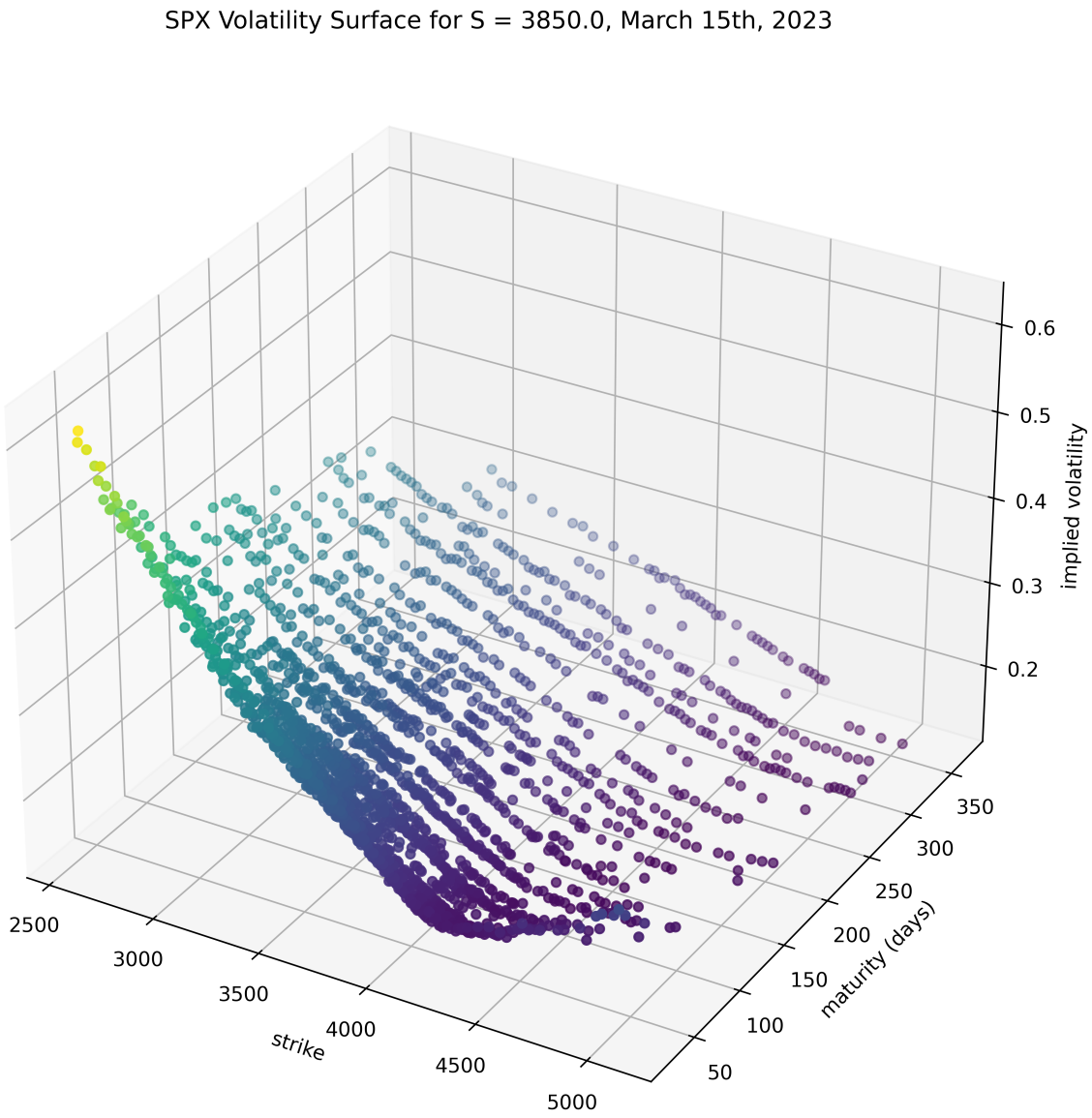


Figure 7: Illustrative set of implied volatilities extracted from one day of SPX options trade data

A.4 Distribution of market parameters

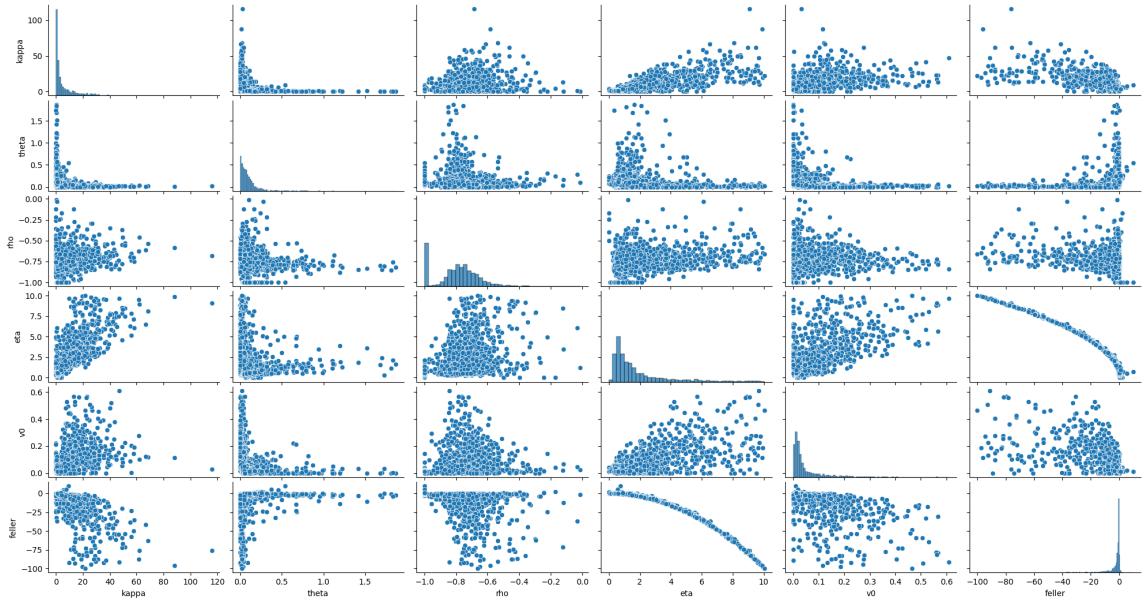


Figure 8: Distribution of Heston [9] pricing model parameters