Pseudo-Analytical Solutions for Stochastic Options Pricing Models using Monte Carlo Simulations and Neural Networks

Samuel Palmer, Denise Gorse

***Abstract*—A combination of Monte-Carlo simulations and neural network learning is used to provide pseudo-analytical solutions for stochastic options pricing models. The neural network is trained to learn the option pricing formula using training samples generated via latin-hyper-cube sampling of Monte-Carlo pricing over the parameter space. Once trained the neural network model has effectively learnt an approximate analytical solution to the problem over the given parameter space. The approximate solution means that unlike other numerical methods it alleviates the need to re-run simulations for different model parameter settings. With extremely large speed and efficiency advantages the neural network models are also shown to produce prices with errors statistically comparable to the Monte-Carlo pricing results presented here. The neural network models were trained to price European call options and Asian call options with arithmetic or geometric averaging.**

I. Introduction

We use artificial neural networks to represent a pseudo-analytical solution of stochastic options pricing models. Using latice-hypercube-sampling (LHS) we sample the option parameter space, for which the sample options are then priced using Monte-Carlo methods. The Monte-Carlo option pricing samples are then used to train an artificial neural network model to approximate the solution of the original stochastic model.

For many cases there does not exist a closed form solution for the stochastic model of financial products such as exotic options, or when using models with more complex model dynamics. One method in these cases is to use simulations of the model using Monte-Carlo (MC) methods [1] to provide the solution. The downfall of MC methods is that for only one parameter set for one stochastic model many thousands of simulation runs may be required. This means that for any change in the parameter values a whole new set of simulations must be ran. This is a common problem for many numerical methods as they do not approximate a complete solution to the model but only provide numerical results for the specific case provided.

As such we propose to use neural networks combined with numerical methods to produce an approximate pseudo-analytical solution for stochastic options pricing models. In essence this methodology uses numerous numerical simulations sampled over the parameter space to train a neural network which then operates as a function approximator over the whole parameter space. Similar techniques have been used in other engineering domains, for example in [2] the authors use neural networks to learn the effect of design parameters over simulated bridge designs.

The training procedure of this method may be in the short-term relatively computationally intensive compared to solving a single parameter setting of a given stochastic model, but in the long term this method can be seen to be extremely efficient as it produces a single neural network model that can be used for all parameter settings over the given parameter space. The solution can be provide in O(1) complexity requiring one simple forward pass through the neural network, which is considerably less computationally expensive than the thousands of simulation runs required for one Monte-Carlo pricing result.

Previous work has used neural networks to learn and predict options prices based on training from actual market data [3] [4] [5]. This may not work for illiquid exotic derivatives/options where there is not enough data. Models trained on market data are also black box solutions with no knowledge of the underlying market models and dynamics, whereas this approach uses well defined stochastic models. In other domains neural networks have been used as approximators to solve differential equations, partial differential equations and stochastic differential equation [6]. Such methods have been used in the finance literature, under the name of meshless methods, to solve models such as the Black-Scholes equation, but this requires the problem to be well defined [7] [8] [9] [10]. As such we look to provide a more flexible novel hybrid numerical method for solving stochastic models.

A. Neural Networks

Artificial neural networks (ANNs) are known as a class of universal approximators inspired by the connectivity of the brain. ANNs consist of simple computing units, known as the 'neurons', connected in a layered structure via numeric weights. Here we use a feed forward multi-layer-perceptron (MLP) network. Traditionally an MLP consists of an input layer, hidden layer/s, and an output layer. Mathematically the output of a individual neurone in a feed forward MLP can be given by

$$y_i^l = f(\sum_i \sum_j w_{i,j}^l y_j^{l-1}) \tag{1}$$

where y_i^l is the output of neuron i in the layer l, when $l = L$ this represents the network output layer and when $l = 0$ this represents the network inputs. Each neurone also has an additional input, $i = 0$, which is known as the bias, this input stays constants for all neurones in all layers and is equal to one. $f(x)$ is known as the activation function of the neuron, the activation function then defines the mapping of the neuron inputs to the neuron output. We use the softplus activation function which is given by

$$f_{softplus}(x) = \log(1 + \exp(x)) \tag{2}$$

; which is commonly used in deep learning applications and was found to be the most effective of the activation functions considered for the given problem.

B. Options Pricing

The most well known model used for pricing option contracts is the geometric brownian motion (GBM) model. In the GMB model the asset price, S, follows a diffusion governed by the following dynamics

$$dS_t = \mu S_t \, dt + \sigma S_t \, dW_t \tag{3}$$

where μ is the mean, σ is the volatility and dW_t is brownian motion.

The defining feature of option contracts is the payoff function, which determines the value of the contract at maturity. European options are the simplest with the payoff given by:

$$P^{\text{call}}_{Euro} = \max(0, K - S_T) \tag{4}$$

where K is the strike price and S_T is the asset price at time of maturity. The analytical price of European options using the brownian motion model is given by the famous Black-Scholes equation [11]. This will be used for comparing the respective errors of the Monte-Carlo prices and neural network model prices.

Other more complex payoff functions can be defined; these are then classed as exotic options. Asian options are popular examples, for Asian options the payoff function, equation 5, is the average of the path values over the option's lifetime, either the geometric average, equation 6, or arithmetic average, equation 7 are used. The closed-form approximation of these two Asian options are given by the Kemna-Vorst approximation [12] and Levy approximation [13] respectively.

$$P^{\text{call}}_{Asian} = \max\left(A(T) - K, 0\right) \tag{5}$$

$$A_{geometric}(T) = \exp\left(\frac{1}{T}\int_0^T \ln(S(t))dt\right) \tag{6}$$

$$A_{arithmetic}(T) = \frac{1}{T}\int_0^T S(t)dt \tag{7}$$

C. Latin Hypercube Sampling

The crux of the proposed methodology relies upon efficiently sampling over the stochastic model parameter space. The issue faced by naive sampling methods, such as grid sampling is that they suffer from the curse of dimensionality and do not scale well with increasing dimensions in the parameter space; as such a random sampling method is required. Latin hypercube sampling [14] is a stratified random sampling method which gives a better distributed representation of the parameter space than just naive random sampling. In LHS each parameter is divided up into equally probable intervals, there is then equal probability that the sample will be chosen from within each interval.

II. Methodology

The proposed pricing method consists of three main stages: 1) generate the options pricing training and validation data via Monte-Carlo (MC) simulations over a range of parameter values; 2) train the neural network to learn the option pricing model from the training data; 3) input desired parameters into the network to obtain price approximations from the neural network model. This is repeated for K number of independent neural network models. We can combine the outputs of each independent neural network model to produce an aggregated neural network model, it was found that using the median output produced the most robust method.

The numerical training data is generated using Monte-Carlo simulations; this method is extremely flexible and can be easily used to price exotic derivatives, but the methodology presented here is not limited to MC and other suitable numerical methods may be used. The MC pricing data has been produced using the Monte-Carlo Longstaff-Schwarz model built into the MatLab Finance toolbox [15], for each pricing run 1000 simulations and 500 periods are used; the corresponding analytical solutions/approximations are also calculated. We sample over the 4D parameter space: interest rate, r; volatility, σ; strike price, k, and asset price, S; these parameters are then used as the inputs to the neural network. The time to maturity is set to a year as this in practice can be changed via the annualised interest rate and volatility in the GBM model.

For each contract type, here we look at European call options and Asian arithmetic and geometric call options, three independent sets of sample data are generated, one for training, one for validation and one for out of sample testing. For each option we generate 2000 samples for training, and 1000 samples for both validation and test data sets. The range of the parameter space used is: $r \in [0.01, 0.1]$; $\sigma \in [0.1, 0.5]$; $S \in [0, 100]$; and $K \in [0, 100]$.

The errors of the numerical approximations $\mathbf{x}$ are compared to the and target data $\mathbf{y}$ using the mean absolute error (MAE) and the mean relative error (MRE) given by

$$E_{MAE}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{N} \text{abs}(x_i - y_i)}{N} \tag{8}$$

$$E_{MRE}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{N} \frac{\text{abs}(x_i - y_i)}{y_i}}{N} \tag{9}$$

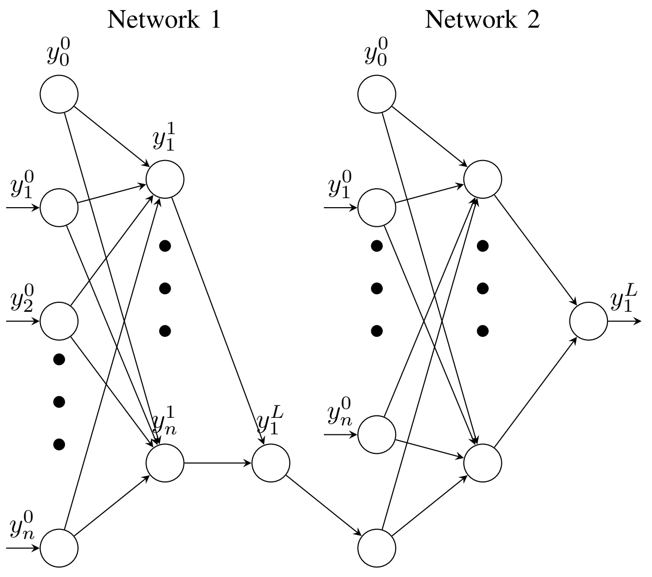


Fig. 1: The architecture of the multi-stage network architecture. This architecture consists of two networks with the output of the first connected as an input to the second, the second network also takes in the original inputs used in network one, the second network then acts as a corrector on the output of the first.

A. Network Architecture

The neural network architecture used is a two step multi-stage network. In this network architecture two smaller networks are connected, where the output/s of the first network and original inputs are both passed in as inputs into the second network. This second network then acts as an additional corrector for errors generated by the first network; a similar network construction was used for example in the successful PSIPRED protein structure predictor [16]. The implemented architecture uses two networks both with two hidden layers of ten neurones.

B. Data Transformations

To aid a network's learning it is often desired to transform the data. Here we apply transforms to both the input parameters and the training target values. Neural network training can be aided by ensuring that the inputs are all of roughly equal magnitude [], the asset price and strike price input parameters are multiplied by 0.01 so they are similar to the magnitudes of the interest rate and volatility. The training target values have a more complex transform applied. The issue with the training data when untransformed is the vast continuous range of magnitudes for prices less than one. Firstly, to reduce the range we apply a minimum resolution by adding a small constant, res, to all the training values. This limits the network to distinguishing values no smaller than res, with this constant small enough it is an appropriate transformation to make. In practice here we test three values of res: 10^{-4}, 10^{-6} and 10^{-8}. The second transform aids the networks learning by transforming the target training values to approximately similar magnitudes, this is done via a $\log_{10}$ type transform

$$T_{sp10}(x) = \log_{10}(10^x - 1)\ ,\ x \neq 0 \tag{10}$$

$$T_{sp10}^{-1}(z) = \log_{10}(10^z + 1) z \neq 0 \tag{11}$$

this is dubbed a softplus-base10 (sp10) transform due to its similarity to the softplus function. The softplus-base10 transform essentially transform all values $x < 1$ using a $\log_{10}$ transform, mapping the x values to negatives values, but remains close to linearity for values $x > 1$. This function is bijective given that for all $x > 0$, and hence is applicable in the case of options pricing which does not involve negative numbers.

The network learns to output the transformed prices; to recover the final price the inverse transform, equation 11, is then applied to the network output.

C. Training Method

The neural networks are trained using the Breeding Particle Swarm Optimisation (BrPSO) algorithm [17]. BrPSO was observed to produce superior neural network training results compared to standard PSO and also gave the best results compared to other algorithms considered in this application. Particle swarm optimisation [18] is a heuristic search method and is based upon the flocking of birds. The algorithm consists of a swarm of particles, for which the position of each particle represents a vector in the search space; in this case the search space is the neural network weights, $\mathbf{W}$. The quality of the position for each particle is evaluated to give a fitness value, for every iteration the particles then move throughout the search space to find the optimum vector.

In this application the fitness value is calculated as a sum of the mean absolute error of the neural network approximations for the transformed option prices and the mean relative error of the inverse transform of the network output compared to the raw training values of the option price, this is given in equation 12. The two component fitness values are used because it was observed when using just the transformed option price small errors in the compressed log transform values resulted in significantly larger errors when the inverse transform was then applied to obtain the final price approximation. When using no transform or training with the inverse transform applied to the network output results were poor. This combination of component allows the network to efficiently output a wide magnitude of prices via the transform but also minimise the respective errors that occur during the inverse transform to the final price. The fitness for each particle in this optimisation can be given by

$$\begin{aligned} \text{fit}(\mathbf{W_i}) = {} & E_{MAE}(N(\mathbf{W_i}, \mathbf{Y^0}), T_{sp10}(\mathbf{V})) \\ & + E_{MRE}(T_{sp10}^{-1}(N(\mathbf{W_i}, \mathbf{Y^0})), \mathbf{V}) \end{aligned} \tag{12}$$

where $\mathbf{W_i}$ is the matrix of neural network weights represented by particle i, $\mathbf{Y^0}$ is the vector of training input parameters sets i.e. $\mathbf{Y^0} = \{\{\mathbf{y_1^0}, \mathbf{y_2^0} ... \mathbf{y_n^0}\}_\mathbf{1}, ... \{\mathbf{y_1^0}, \mathbf{y_2^0} ... \mathbf{y_n^0}\}_\mathbf{J}\}$, $\mathbf{V}$ is the corresponding vector of target prices for input parameter sets, and $N(\mathbf{W}, \mathbf{Y^0})$ is the vector of neural network approximation outputs for each input parameter set given in $\mathbf{Y^0}$.

III. Results and Discussion

Results are presented for the neural network price approximations for Europan call options, and Asian arithmetic and geometric call options. The methodology discussed in the preceding section is repeated for each of the options contracts, with 30 neural networks being trained for each contract.

For European options it is seen that the aggregated neural network model, using the median output of the trained neural networks, produces far superior results to an individual network, as well as being a more robust and reliable solution. As such only the results for the aggregated neural network model will be presented for the Asian options. It should be noted that only one aggregated neural network model is created for each contract using the individually trained neural networks.

A. European Options

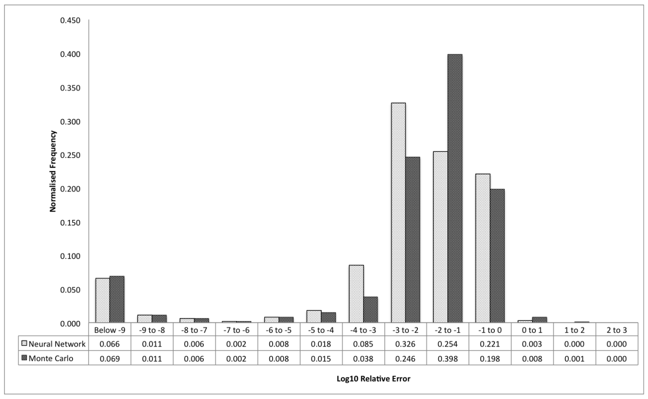
The error results for the independent neural network models are given in table I, and for the aggregated neural network model in table II. When using the independent neural networks for $res = 10^{-8}$ only 28 networks were analysed, this is because two networks presented infinite results for two option prices; this is due to overfitting in the network training, validation was used to reduce the effects of overfitting but as seen it is not always successful. As such this shows that using a single independent neural network may not lead to reliable solution. Even for $res = 10^{-4}$ and 10^{-6} where there were no errors in the neural network results the errors are still significantly larger than using the aggregated neural network models. Even though some independent neural network models may produce errors lower than the aggregated model the variance makes selecting a model not as reliable as supposed to using an aggregated model. It can be observed that the aggregated model, as expected, provides more consistent results than using a single neural network model

In all cases the MAE for the aggregated neural network models are significantly better than the Monte-Carlo results, being up to two times smaller. On the other hand the MREs are slightly larger; from this combined with the lower MAE it can be inferred that the aggregated neural network is more accurate for estimating the price of options with prices of a larger magnitude, i.e. in-the-money options. Although the lower MRE then suggests that for prices of a lower magnitude, i.e. out-of-the-money options, the neural network may be slightly less accurate, and this an area further development of the methodology can improve. The reason for the MRE for lower magnitude values is due to the logarithmic type data transform applied to these values during training; the compression of the range of target values using the log transform means that during training small training errors can result in larger errors once the inverse transform is applied. Even though precautions were taken to minimise this effect by using a two component fitness function involving the inverse-transform MRE the effect is still observable.

In addition we have looked at the probability of the relative error being less than 10%, $P_{RE}(< 10\%)$. For all of the cases it can be seen that the aggregated neural network has a similar probability in these examples with only a maximum of 2% lower probability; in fact for $res = 10^{-6}$ the probability for the aggregated neural network model is slightly higher. Figures 2, 3 and 4 show the distribution of the magnitudes of the relative errors, it can actually be seen that in all cases the neural network model has a higher proportion of errors with magnitude of negative two or lower, which corresponds to percentage error of $< 1\%$. For both $res = 10^{-6}$ and 10^{-8} there are no relative errors greater than 100% whilst the Monte Carlo results have a small proportion for all resolution values.

For Monte-Carlo we see a steady decreases in $P_{RE}(< 10\%)$ as the resolution constant decreases, but for the neural network we see that there is an increases for when $res = 10^{-6}$. This inconsistancy may suggest that the current aggregated neural network results have not converged to a stable distributions and have further room for improvement. The results presented here only used an aggregation of 30 independent neural networks and it is hoped that results can be further improved by using more.

It can be seen that overall the numerical results presented suggest that a suitable pseudo-analylitical solution using neural networks can be obatined for pricing European options. The errors of the aggregated neural network model are comparable and in some cases better than the Monte-Carlo results for pricing European options; but also with the advantage that the neural network model price approximations can be generated innumerably faster and efficiently. This methodology is then tested on the more complex case of Asian option contracts.



	Below -9	-9 to -8	-8 to -7	-7 to -6	-6 to -5	-5 to -4	-4 to -3	-3 to -2	-2 to -1	-1 to 0	0 to 1	1 to 2	2 to 3
▨ Neural Network	0.066	0.011	0.006	0.002	0.008	0.018	0.085	0.326	0.254	0.221	0.003	0.000	0.000
▩ Monte Carlo	0.069	0.011	0.006	0.002	0.008	0.015	0.038	0.246	0.398	0.198	0.008	0.001	0.000

Fig. 2: Histogram showing the distribution of the magnitudes of mean relative errors for European call options, with $res = 10^{-4}$.

B. Asian Options

The error results for the Asian geometric and arthiemtic aggregated neural network models are presented in tables III and IV. Both Asian options actually seem to be easier to approximate than European options, with the MREs and $P_{RE}(< 10\%)$ values being lower, this is also true for the MC simulations.

This can be seen with a larger difference between the $P_{RE}(< 10\%)$ values for the MC and neural network price estimations, the neural network approximations are seen to have a $4 - 5\%$ lower probability, whilst this is only $1 - 2\%$

	MAE Mean	MAE StDev	MRE Mean	MRE StDev	$P_{RE}(< 10\%)$ Mean	$P_{RE}(< 10\%)$ StDev
$res = 10^{-4}$	0.206	0.034	0.173	0.014	0.750	0.017
$res = 10^{-6}$	0.245	0.080	0.203	0.033	0.724	0.036
$res = 10^{-8}$	0.277	0.074	0.244	0.037	0.687	0.036

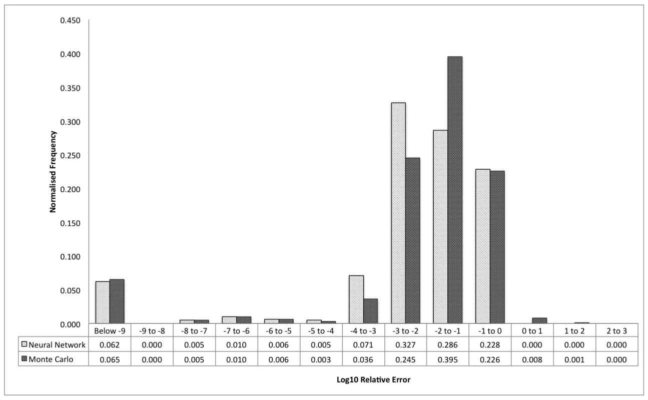
TABLE I: The mean absolute error (MAE), mean relative error (MRE) and probability of relative error being less than 10% ($P_{RE}(< 10\%)$) for European call options test set. The mean and standard deviation values of the error measures of the independent neural network models.

	MAE	MAE StDev	MedAE	MRE	MRE StDev	MedRE	$P_{RE}(< 10\%)$
$res = 10^{-4}$							
MC	0.295	0.446	0.130	0.128	0.616	0.017	0.793
Ag-NNM	0.131	0.140	0.095	0.161	0.325	0.008	0.776
$res = 10^{-6}$							
MC 1.00E-08	0.295	0.446	0.130	0.185	0.731	0.023	0.742
Ag-NNM	0.165	0.197	0.110	0.232	0.385	0.016	0.722
$res = 10^{-8}$							
MC	0.295	0.446	0.130	0.159	0.717	0.019	0.765
Ag-NNM	0.137	0.165	0.089	0.181	0.345	0.011	0.772

TABLE II: The mean absolute error (MAE), median absolute error (MedAE), mean relative error (MRE), median relative error (MedRE) and the probability of relative error being less than 10% ($P_{RE}(< 10\%)$) for the European call options test data set. Results are for both the aggregated neural network model (Ag-NNM) and Monte-Carlo (MC) price approximations for European options.

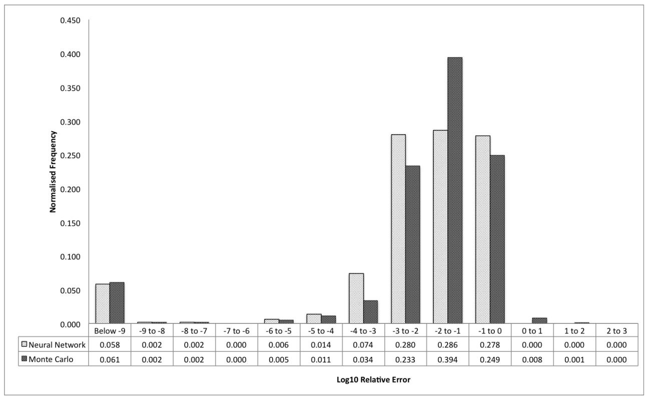
	MAE	MAE StDev	MedAE	MRE	MRE StDev	MedRE	$P_{RE}(< 10\%)$
$res = 10^{-4}$							
MC	0.053	0.088	0.015	0.153	1.409	0.002	0.832
Ag-NNM	0.091	0.133	0.046	0.151	0.304	0.003	0.780
$res = 10^{-6}$							
MC	0.053	0.088	0.015	0.262	3.285	0.002	0.806
Ag-NNM	0.101	0.163	0.043	0.210	0.377	0.004	0.754
$res = 10^{-8}$							
MC	0.053	0.088	0.015	0.284	3.332	0.003	0.788
Ag-NNM	0.099	0.164	0.052	0.223	0.390	0.006	0.740

TABLE III: The mean absolute error (MAE), median absolute error (MedAE), mean relative error (MRE), median relative error (MedRE) and the probability of relative error being less than 10% ($P_{RE}(< 10\%)$) for the Asian geometric average call options test data set. Results are for both the aggregated neural network model (Ag-NNM) and Monte-Carlo (MC) price approximations for Asian geometric options.



	Below -9	-9 to -8	-8 to -7	-7 to -6	-6 to -5	-5 to -4	-4 to -3	-3 to -2	-2 to -1	-1 to 0	0 to 1	1 to 2	2 to 3
▨ Neural Network	0.062	0.000	0.005	0.010	0.006	0.005	0.071	0.327	0.286	0.228	0.000	0.000	0.000
■ Monte Carlo	0.065	0.000	0.005	0.010	0.006	0.003	0.036	0.245	0.395	0.226	0.008	0.001	0.000

Fig. 3: Histogram showing the distribution of the magnitudes of mean relative errors for European call options, with $res = 10^{-6}$.



	Below -9	-9 to -8	-8 to -7	-7 to -6	-6 to -5	-5 to -4	-4 to -3	-3 to -2	-2 to -1	-1 to 0	0 to 1	1 to 2	2 to 3
▨ Neural Network	0.058	0.002	0.002	0.000	0.006	0.014	0.074	0.280	0.286	0.278	0.000	0.000	0.000
■ Monte Carlo	0.061	0.002	0.002	0.000	0.005	0.011	0.034	0.233	0.394	0.249	0.008	0.001	0.000

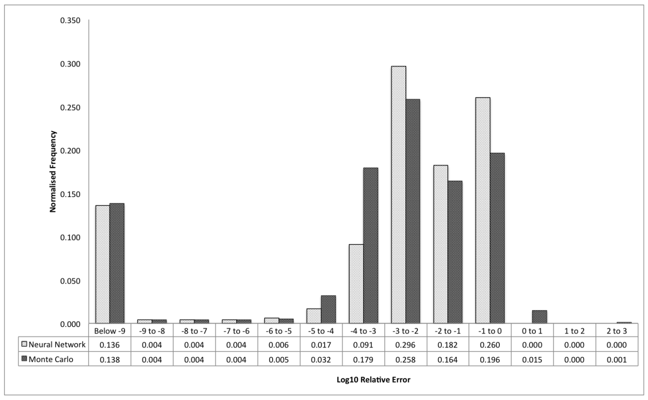
Fig. 4: Histogram showing the distribution of the magnitudes of mean relative errors for European call options, with $res = 10^{-8}$.

	MAE	MAE StDev	MedAE	MRE	MRE StDev	MedRE	$P_{RE}(< 10\%)$
$res = 10^{-4}$							
MC 1.00E-04	0.063	0.107	0.017	0.277	3.106	0.002	0.837
Ag-NNM	0.087	0.128	0.039	0.146	0.316	0.003	0.786
$res = 10^{-6}$							
MC 1.00E-06	0.063	0.107	0.017	0.262	3.285	0.002	0.811
Ag-NNM	0.101	0.163	0.043	0.191	0.361	0.004	0.770
$res = 10^{-8}$							
MC	0.063	0.107	0.017	0.435	5.542	0.003	0.794
Ag-NNM	0.090	0.139	0.043	0.211	0.381	0.005	0.744

TABLE IV: The mean absolute error (MAE), median absolute error (MedAE), mean relative error (MRE), median relative error (MedRE) and the probability of relative error being less than 10% ($P_{RE}(< 10\%)$) for the Asian arithmetic average call options test data set. Results are for both the aggregated neural network model (Ag-NNM) and Monte-Carlo (MC) price approximations for Asian arithmetic options.

for European options. Figures **??** and **??** give the histograms of the relative errors for the Asian geometric and arithmetic options respectively. It can be seen that whilst the neural networks have a higher proportion of errors between 0.1% - 10%, MC does produce a higher proportion of results below 0.1%. Unfortunately the neural network model does have a significantly higher proportion of results between 10% - 100%, this is due to the network underestimating options prices.When underestimating the option price the relative error is bounded by one (100%), and during optimisation this is desirable to minimise the MRE, as such this may skew training away from focusing on better learning option prices which fall into this criteria. Though it can be seen that again the MC does produce approximations with relative errors larger than 100% where the price has been overestimated, whilst the neural network model does not produce any overestimations.

Overall the neural network approximations are still very close to the accuracy of Monte-Carlo with a high probability of low relative errors. This accuracy combined with the incredibly large speed up makes this method advantageous to use compared to Monte-Carlo used here.

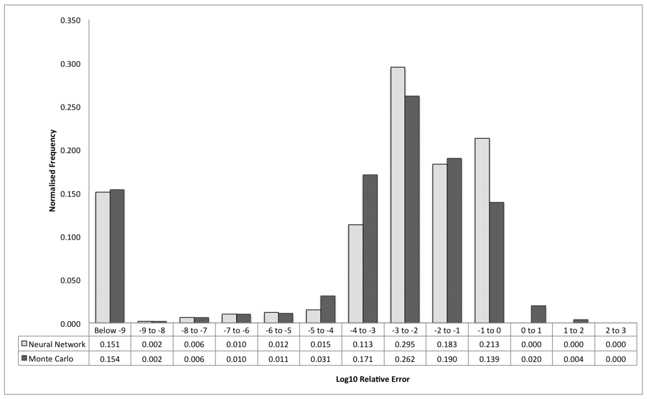


	Below -9	-9 to -8	-8 to -7	-7 to -6	-6 to -5	-5 to -4	-4 to -3	-3 to -2	-2 to -1	-1 to 0	0 to 1	1 to 2	2 to 3
▨ Neural Network	0.136	0.004	0.004	0.004	0.006	0.017	0.091	0.296	0.182	0.260	0.000	0.000	0.000
■ Monte Carlo	0.138	0.004	0.004	0.004	0.005	0.032	0.179	0.258	0.164	0.196	0.015	0.000	0.001

Fig. 5: Histogram showing the distribution of the magnitudes of mean relative errors for Asian geometric average call options, with $res = 10^{-8}$.

IV. Conclusion

We present a novel hybrid methodology using traditional numerical methods and neural networks to produce a pseudo-



	Below -9	-9 to -8	-8 to -7	-7 to -6	-6 to -5	-5 to -4	-4 to -3	-3 to -2	-2 to -1	-1 to 0	0 to 1	1 to 2	2 to 3
▨ Neural Network	0.151	0.002	0.006	0.010	0.012	0.015	0.113	0.295	0.183	0.213	0.000	0.000	0.000
■ Monte Carlo	0.154	0.002	0.006	0.010	0.011	0.031	0.171	0.262	0.190	0.139	0.020	0.004	0.000

Fig. 6: Histogram showing the distribution of the magnitudes of mean relative errors for Asian arithmetic average call options, with $res = 10^{-8}$.

analytical solution for stochastic options pricing models. The results presented here show that when compared to the analytical solutions/approximations the aggregated neural network model produces prices that have errors similar to the Monte-Carlo method used. The advantage of this method is that unlike Monte-Carlo or other numerical methods the neural network represents an analytical approximation of the solution over the whole parameter space. This means that neural network model only needs to be trained once, and once trained can evaluate option prices extremely fast over the parameter space.

We have tested this methodology on simple European options and the more complex case of Asian options, but only using simple stochastic models. It will be interesting to see how this methodology fairs with more complex stochastic models, for example pricing options using the Heston model [19] which incorporates stochastic volatility, or further using the Heston-Hull-White model [20] which has both stochastic volatility and interest rates. Apart from using the networks to calculate price approximations the models could be further investigated by analysing the greek sensitivity values of the options using the neural network models. It will also be interesting to see how sensitive the neural network training is to the accuracy of the underlying numerical training data; it could be suggested that more simulations are required for the

Monte-Carlo results here.

There are some limitations to the current method, mainly being the pricing results for out-of-the-money options for which the prices are respectively many magnitudes smaller than in-the-money options. The problem due to the wide range of magnitudes of price outputs was partially resolved by using the softplus-base10 transform, equation 10 suggested in this work; however further work will be needed to improve the relative errors in these instances.

Overall this work presents a successful exploratory study into using neural networks as a method of representing pseudo-analytical approximations for stochastic option pricing models. The method produces results with good accuracy and has the advantage of extremely efficient price evaluations when compared to more traditional numerical methods. With further work there is scope to further increase the accuracy and efficiency of this methodology.

References

[1] P. Glasserman, *Monte Carlo methods in financial engineering*, vol. 53. Springer Science & Business Media, 2003.

[2] D. Lehký and M. Šomodíková, *Engineering Applications of Neural Networks: 16th International Conference, EANN 2015, Rhodes, Greece, September 25-28 2015.Proceedings*, ch. Reliability Analysis of Post-Tensioned Bridge Using Artificial Neural Network-Based Surrogate Model, pp. 35–44. Cham: Springer International Publishing, 2015.

[3] J. Bennell and C. Sutcliffe, "Black–scholes versus artificial neural networks in pricing ftse 100 options," *Intelligent Systems in Accounting, Finance and Management*, vol. 12, no. 4, pp. 243–260, 2004.

[4] B. K. Wong and Y. Selvi, "Neural network applications in finance: A review and analysis of literature (1990–1996)," *Information & Management*, vol. 34, no. 3, pp. 129–139, 1998.

[5] U. Anders, O. Korn, and C. Schmitt, "Improving the pricing of options: A neural network approach," tech. rep., ZEW Discussion Papers, 1996.

[6] N. Yadav, A. Yadav, and M. Kumar, *An Introduction to Neural Network Methods for Differential Equations*. Springer, 2015.

[7] Y. Goto, Z. Fei, S. Kan, and E. Kita, "Options valuation by using radial basis function approximation," *Engineering Analysis with Boundary Elements*, vol. 31, no. 10, pp. 836–843, 2007.

[8] A. Golbabai, D. Ahmadian, and M. Milev, "Radial basis functions with application to finance: American put option under jump diffusion," *Mathematical and Computer Modelling*, vol. 55, no. 3, pp. 1354–1362, 2012.

[9] O. González-Gaxiola and P. P. González-Pérez, "Nonlinear black-scholes equation through radial basis functions," *Journal of Applied Mathematics and Bioinformatics*, vol. 4, no. 3, p. 75, 2014.

[10] Y.-C. Hon and X.-Z. Mao, "A radial basis function method for solving options pricing models," *Journal of Financial Engineering*, vol. 8, pp. 31–50, 1999.

[11] J. B. Cohen, F. Black, and M. Scholes, "The valuation of option contracts and a test of market efficiency," *The Journal of Finance*, vol. 27, no. 2, pp. 399–417, 1972.

[12] A. G. Kemna and A. Vorst, "A pricing method for options based on average asset values," *Journal of Banking & Finance*, vol. 14, no. 1, pp. 113–129, 1990.

[13] E. Levy, "Pricing european average rate currency options," *Journal of International Money and Finance*, vol. 11, no. 5, pp. 474–491, 1992.

[14] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.

[15] "Matlab and finance toolbox release 2014b, the mathworks, inc, natick, massachusetts, united states.,"

[16] L. J. McGuffin, K. Bryson, and D. T. Jones, "The psipred protein structure prediction server," *Bioinformatics*, vol. 16, no. 4, pp. 404–405, 2000.

[17] S. Palmer, D. Gorse, and E. Muk-Pavic, "Neural networks and particle swarm optimization for function approximation in tri-swach hull design," in *Proceedings of the 16th International Conference on Engineering Applications of Neural Networks (INNS)*, p. 8, ACM, 2015.

[18] R. C. Eberhart, J. Kennedy, *et al.*, "A new optimizer using particle swarm theory," in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1, pp. 39–43, New York, NY, 1995.

[19] S. L. Heston, "A closed-form solution for options with stochastic volatility with applications to bond and currency options," *Review of financial studies*, vol. 6, no. 2, pp. 327–343, 1993.

[20] L. A. Grzelak, C. W. Oosterlee, and S. Van Weeren, "Extension of stochastic volatility equity models with the hull–white interest rate process," *Quantitative Finance*, vol. 12, no. 1, pp. 89–105, 2012.