Pseudo-Analytical Solutions for Stochastic Options Pricing Using Monte Carlo Simulation and Breeding PSO-Trained Neural Networks

Sam Palmer and Denise Gorse

University College London - Dept of Computer Science
Gower Street, London, WC1E 6BT - UK

Abstract. We introduce a novel methodology for pricing options which uses a particle swarm trained neural network to approximate the solution of a stochastic pricing model. The performance of the network is compared to the analytical solution for European call options and the errors shown statistically comparable to Monte Carlo pricing. The work provides a proof of concept that can be extended to more complex options for which no analytical solutions exist, the pricing method presented here delivering results several orders of magnitude faster than the Monte Carlo pricing method used by default in the financial industry.

1 Introduction

Option contracts have been bought and sold for hundreds of years. They originated in the agricultural sector as a means by which farmers could fix a price for the sale of commodities such as corn and wheat, in advance of a harvest which might be unpredictably affected by the weather. Options are nowadays used not only for the hedging of risk but also as vehicles for speculative investment, and their pricing is an important problem to which machine learning methods, including neural networks (see for example [1] and references therein), are being increasingly applied.

In this work we use neural networks (NNs), trained with a form of particle swarm optimisation, Breeding PSO (BrPSO), which has shown itself in previous work [2] to be adept for this purpose, to produce an approximate pseudo-analytical solution for stochastic options pricing models. Once trained, such a network can be used for all parameter settings to generate the corresponding option price estimate with O(1) complexity, compared to the thousands of simulation runs required for a single Monte Carlo pricing result. Similar techniques have been used in other domains, for example in [3] neural networks learn the effect of design parameters on simulated bridge designs.

2 Financial background

An option gives its owner the right, but not obligation, to trade a certain amount of an underlying asset at some future time. A *call option* gives its owner the right to buy that amount of the asset, a *put option* the right to sell. In the simplest case, referred to as a *European option*, the time in question is the date on which the contract matures; however many more complex financial derivative products have been devised.

In the case of a European call option there is a well-known analytical solution to the pricing problem, known as the Black-Scholes (B-S) model [4]. The inputs to this model are the risk-free interest rate, r; the volatility in the price of the underlying asset, σ; the *strike price* (agreed price at maturity), k, and the current asset price, s; the output of the model is a value for the price of the option at a time t to maturity.

However, as mentioned in the introduction, for more complex derivative products such an analytical solution for the stochastic model that describes the pricing process does not exist. In this case it is necessary to employ Monte Carlo (MC) simulation [5] or some equivalent stochastic modelling process. Such processes demand thousands of time-consuming runs in order to get a reliable estimate of a fair price for the option, which is why we here aim to use PSO-trained neural networks to replace the use of Monte Carlo simulation.

Much work has been carried out in the application of machine learning methodologies to options pricing. However in most cases the training is based on market data (as for example in [6], [7]) which may not work for illiquid exotic derivative products where there is not enough data. In addition models trained on market data are 'black box' solutions with no knowledge of the underlying market models and dynamics, and as such may be less attractive to financial institutions in the current regulatory climate. In contrast the approach to be taken here seeks to use a neural network to approximate the behaviour of a well defined stochastic model, and to our knowledge is the first deployment of such an approach in the financial domain.

3 Methodology

We first generate the options pricing training and validation data via Monte-Carlo simulations over a range of parameter values. A PSO-trained neural network then learns the option pricing model from these data. K=30 independently trained neural networks are combined to produce an aggregated neural network model; it was found that using the median output produced the most robust results, and it is from such an aggregated model that the out-of-sample test prices are generated.

3.1 Data generation

As mentioned earlier the training data for these experiments is generated using Monte Carlo simulation; this method is very flexible and can be used to price exotic derivatives, but the methodology presented here is not limited to data generated via MC and other numerical methods may be used. The MC pricing data is produced using the Monte-Carlo Longstaff-Schwarz model built into the MatLab Finance toolbox [8]; for each run 1000 simulations and 500 periods are used. We sample over the 4D parameter space defined earlier for the B-S model, in the ranges $r \in [0.01, 0.1]$; $\sigma \in [0.1, 0.5]$; and $k, s \in [0, 100]$. Three independent sets of data are generated: 2000 samples for training, and 1000 samples for each of validation and out-of sample testing. Latin hypercube sampling (LHS) [9] is used in preference to either grid or naïve random sampling as LHS scales less badly than the former while giving a better distributed representation of the parameter space than the latter.

3.2 I/O data transformations

It was found that some data transformations were required to make the task of training the neural network tractable. 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 that aims to better separate otherwise overly-close targets. First, to reduce the range, we apply a minimum resolution by adding a small constant, *res*, to all the target values, limiting the network to distinguishing values no smaller than *res*; we test two values of *res*, 10^{-6} and 10^{-8}. Second, we further aid learning by transforming the target values, which could otherwise differ by many orders of magnitude, to approximately similar magnitudes via the introduction of a novel transform, $T_{sp10}(x)=\log_{10}\left(10^{x}-1\right), \quad x\neq 0$, dubbed by us a *softplus-base*10 (*sp*10) transform due to its similarity to the softplus function used in the nets' hidden layers. Softplus-base10 is bijective provided $x > 0$ and hence is applicable in the case of options pricing as this does not involve negative numbers. The network learns to output the transformed prices; to recover the price the inverse transform is applied to the network output. It is important to note that without the sp10 transformation the problem would be intractable; many alternative output transforms were considered but our sp10 transform was by far the most effective.

3.3 Neural network architecture

The neural network architecture consists of two connected networks, both with two hidden layers of ten neurons, in which the output of the first network and the original inputs are passed on as inputs to the second network. This second network then acts as a corrector for errors generated by the first network; a similar network construction was used, for example, in the successful PSIPRED protein structure predictor [10], and provides superior results compared to the use of only a single network. The hidden layer activation function for both networks is softplus, defined as $f_{softplus}(x)=\log(1+\exp(x))$; this function is commonly used in deep learning applications and was the most effective of the activation functions considered here.

3.4 Training method

The neural networks are trained using Breeding Particle Swarm Optimisation (BrPSO) [2], a new form of PSO developed by one of us (SP) which displays enhanced search capabilities via the use of a particle swarm that evolves via natural selection in a way inspired by the Comprehensive Learning Particle Swarm Optimizer (CLPSO) [11] and Global-Local Differential Evolution (GLDE) [12] algorithms. BrPSO was picked here because of its performance in [2] when benchmarked against competitor neural network training algorithms. The current work uses a swarm of 100 particles, the PSO algorithms being run for 5000 iterations. Social and cognitive learning contributions are weighted equally at 2.05, with a constriction factor equal to 0.7289. These are default parameter choices in the PSO literature; experimentally it has been found that their manipulation produces little benefit and introduces an unnecessary additional risk of overfitting. All BrPSO-specific parameter settings as in [2], 10 runs being performed under each set of experimental conditions. In any form of PSO the quality of the position for each particle *i* must be evaluated to give a fitness value which is used to guide the progress of the particles toward the optimum parameter settings (NN weights $\mathbf{W}_i$). Fitness is here given by

$$\mathrm{fit}(\mathbf{W}_i)=E_{MAE}\left(N\left(\mathbf{W}_i,\mathbf{Y}^{\mathbf{0}}\right),T_{sp10}(\mathbf{V})\right)+E_{MRE}\left(T^{-1}_{sp10}(N(\mathbf{W}_i,\mathbf{Y}^{\mathbf{0}})),\mathbf{V}\right),$$

where $\mathbf{W}_i$ is the matrix of neural network weights represented by particle *i*, $\mathbf{Y}^{\mathbf{0}}$ is the matrix whose rows are the four-dimensional training input parameter vectors, $\mathbf{V}$ is the corresponding vector of target prices for those input parameter sets, $N(\mathbf{W}_i,\mathbf{Y}^{\mathbf{0}})$ is the vector of neural network-approximated outputs, and where $E_{MAE}(\mathbf{x},\mathbf{y})$ and $E_{MRE}(\mathbf{x},\mathbf{y})$ are the mean absolute and mean relative errors, respectively, of the numerical approximations $\mathbf{x}$ compared to targets $\mathbf{y}$, given by

$$E_{MAE}(\mathbf{x},\mathbf{y})=\frac{1}{N}\sum_{i=1}^{N}|x_i-y_i|\ ,\quad E_{MRE}(\mathbf{x},\mathbf{y})=\frac{1}{N}\sum_{i=1}^{N}|x_i-y_i|\,/y_i\ .$$

This sum of mean absolute and mean relative errors encourages the network to output a wide magnitude range of prices while at the same time minimising the errors that would otherwise occur during the inverse transform to a final price.

4 Results

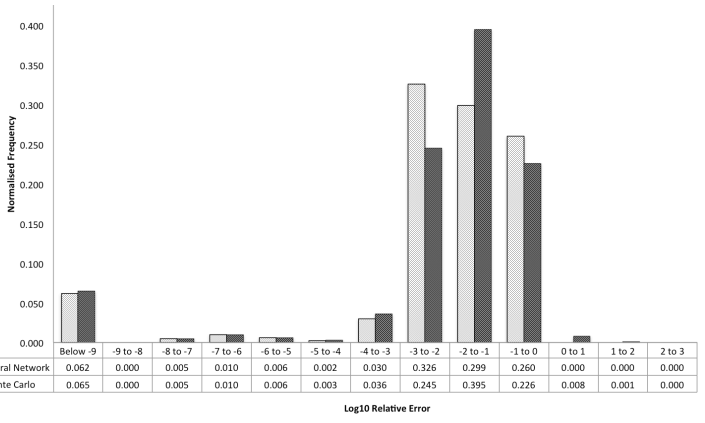
The out-of-sample test results for European call options are shown in Table 1. For such options, as discussed above, there is an analytical solution (the Black-Scholes model) to which both the Monte Carlo and NN/PSO pricing methodologies can be compared, for two settings of the input value resolution bound *res*.

res	method	MAE±StDev	MedAE	MRE±StDev	MedRE	P_{RE}(<10%)	P_{RE}(<10%)
10^{-6}	MC	0.295±0.446	0.130	0.128±0.616	0.023	0.765	0.370
	NN/PSO	0.160±0.168	0.116	0.204±0.361	0.014	0.740	0.441
10^{-8}	MC	0.295±0.446	0.130	0.159±0.717	0.019	0.742	0.348
	NN/PSO	0.165±0.176	0.115	0.228±0.381	0.017	0.716	0.410

Table 1: Mean absolute (MAE), median absolute (MedAE), mean relative (MRE), and median relative (MedRE) test errors, and the probabilities of the relative error being less than 10% (P_{RE}(< 10%) and less than 1% P_{RE}(< 1%)). Results are presented for both the BrPSO-trained aggregated neural network model with K=30 (PSO/NN) and Monte-Carlo (MC) price approximations.

As can be seen above in all cases the MAE for the NN/PSO model is significantly better than the Monte-Carlo result, being up to two times smaller. On the other hand the MREs are slightly larger than could be obtained from an MC simulation; from this, combined with the lower MAEs, it can be inferred that the NN/PSO method is less accurate for estimating the price of options which have a larger magnitude, i.e. are *out-of-the-money* options (in the case of call options, ones for which the strike price is above the market price). The reason MREs in such cases are higher is connected to the necessary transform applied to target values during training; the output range compression means that small training errors can result in larger errors when the inverse transform is applied during use. Though precautions were taken to minimise this effect by using a two component fitness function (Eq. 2) the effect is still observable and further improvements to the methodology will be sought.

In addition we have looked at the probability of the relative test error being less than 10%, P_{RE} (< 10%), and less than1%, P_{RE}(<1%). We can see that compared to MC the NN/PSO P_{RE} (<10%) is only slightly lower, but that our method displays a sizeably higher P_{RE} (<1%). This can also be observed in Figure 1 which shows the distribution of the magnitudes of the relative test errors. It can be seen in this figure that in addition to offering a speed-up, in online post-training use, of at least three orders of magnitude in comparison to running Monte Carlo simulations, the NN/PSO method also compares well in terms of accuracy, displaying considerably fewer errors than MC with magnitudes in the range from 10^{-3} to 10^{-2} (this range corresponding to a percentage error of around 1%). It should also be noted that while the Monte Carlo model occasionally generates relative errors greater than 100%, the NN/PSO results include no instances of such unacceptable performance. We note finally that the results here use an aggregation of only K=30 independent neural networks; it is hoped that results can be further improved by using more networks in the committee.



	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	0.002	0.030	0.326	0.299	0.260	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. 1: Histogram showing magnitude distribution of the mean relative errors for European call options with *res*=10^{-8} (dark: Monte Carlo; light: NN/PSO).

5 Discussion

This work has presented a novel hybrid methodology that uses both traditional numerical methods and neural networks to produce a pseudo-analytical solution for stochastic options pricing models. This work provides a proof of concept, in a case for which an analytical solution (the Black-Scholes equation) is available, and to which both the NN/PSO and Monte Carlo results can be compared, that an NN/PSO method could be used as a faster alternative method for the pricing of derivative products for which no closed form solutions are available and for which the only recourse would otherwise be to the use of time consuming Monte Carlo simulations.

More detailed analysis of the results shows that for *deep-in-the-money* options (in the case of call options, ones for which the strike price is significantly below the market price) the NN/PSO results are more accurate than the MC prices; in fact for all such deep-in-the-money instances in the test set the mean relative error is always less than 1%, which is not the case observed for the MC prices. There are some limitations to the current method, mainly the less accurate pricing of out-of-the-money options. The underlying problem was resolved in part by a combination of the softplus-base10 transform (Eq. 1) and two-component fitness function (Eq. 2); however more work will be needed to improve performance for out-of-the-money options.

Overall we consider this work a successful exploratory study, producing results with good accuracy and with the advantage of extremely efficient price evaluation when compared to more traditional numerical methods. We intend to carry out work to further increase the accuracy and efficiency of this methodology before applying it to the pricing of more complex financial derivative products.

References

[1] J.T. Hahn, Option pricing using artificial neural networks: An Australian perspective. Ph.D. Thesis, Faculty of Business, Bond University, Robina, Queensland 4226, Australia, November 2013.

[2] S. Palmer, D. Gorse and E. Muk-Pavic, Neural networks and particle swarm optimization for function approximation in Tri-SWACH hull design. In L. Iliadis and C. Jayne, editors, proceedings of the *16th International Conference on Engineering Applications of Neural Networks* (EANN 2015), Article No. 8, Springer International Publishing, 2015.

[3] D. Lehký and M. Šomodíková, Reliability analysis of post-tensioned bridge using artificial neural network-based surrogate model, *Communications in Computer and Information Science*, 517: 35-44, Springer International Publishing, 2015.

[4] F. Black and M.S. Scholes, The pricing of options and corporate liabilities, *Journal of Political Economy*, 81(3):637-654, Univ. of Chicago Press, 1973.

[5] P. Glasserman, *Monte Carlo Methods in Financial Engineering*, vol. 53, Springer Science & Business Media, New York, 2003.

[6] C. Von Spreckelsen, H.-J. Von Mettenheim and M.H. Breitner, Real-time pricing and hedging of options on currency futures with artificial neural networks, *Journal of Forecasting*, 33:419–432, Wiley, 2014.

[7] Y. Lin and J. Yang, Option pricing model based on Newton-Raphson iteration and RBF neural network using implied volatility, *Canadian Social Science*, 12(8):25-29, Canadian Academy of Oriental and Occidental Culture, 2016.

[8] Matlab Finance Toolbox release 2014b. The Mathworks, Inc., Natick, MA, USA.

[9] 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*, 42(1):55–61, American Statistical Association, 2000.

[10] L.J. McGuffin, K. Bryson and D.T. Jones, The PSIPRED protein structure prediction server, *Bioinformatics*, 16(4):404–405, Oxford Univ. Press, 2000.

[11] J.J. Liang, A.K. Qin and S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, *IEEE Trans. Evolutionary Computation*, 10(3):281-295, IEEE, 2006.

[12] S. Das, A. Abraham, U.K. Chakraborty and A. Konar, Differential evolution using a neighborhood-based mutation operator, *IEEE Trans. Evolutionary Computation*, 13(3):526-553, IEEE, 2009.