NN-OPT: Neural Network for Option Pricing Using Multinomial Tree

Hung-Ching (Justin) Chen and Malik Magdon-Ismail

Rensselaer Polytechnic Institute, Dept. of Computer Science, Troy, NY 12180, USA
{chenh3, magdon}@cs.rpi.edu

Abstract. We provide a framework for learning to price complex options by learning risk-neutral measures (Martingale measures). In a simple geometric Brownian motion model, the price volatility, fixed interest rate and a no-arbitrage condition suffice to determine a unique risk-neutral measure. On the other hand, in our framework, we relax some of these assumptions to obtain a *class* of allowable risk-neutral measures. We then propose a framework for learning the appropriate risk-neural measure. In particular, we provide an efficient algorithm for backpropagating gradients through multinomial pricing trees. Since the risk-neutral measure prices all options simultaneously, we can use all the option contracts on a particular stock for learning. We demonstrate the performance of these models on historical data. Finally, we illustrate the power of such a framework by developing a real time trading system based upon these pricing methods.

1 Introduction

In 1973, Black and Scholes published their pioneering paper [1] which introduced the first option pricing formula and also developed a general framework for derivative pricing. Since then, derivative pricing has become a popular research topic. A modern, popular approach to pricing has been though the Martingale measure (see, for example, [2]). The origin of the fundamental theorems on the Martingale measure can be traced to Cox and Ross' paper [3] describing the method of *risk neutral valuation*. The Martingale measure was developed into a more mature pricing technique in [4–7]. Other related topics can be found in [2, 8].

Option trading by directly predicting prices and then building trading systems based on the predictions have been considered in the neural network literature [9, 10]. An alternative to predicting prices and then trading is to use direct reinforcement to trade directly (see for example [11]). Learning to trade directly has the advantage of avoiding an additional price-prediction step. When multiple instruments are available, for example multiple options on a single underlying stock, then the state space of possible trading actions grows exponentially and direct reinforcement for learning to trade becomes infeasible. In addition, price prediction of each individual option leads to an excessive number parameters, and it now makes sense to develop a unified price prediction mechanism for all

the options simultaneously. Once prices are predicted for all the options, trading can be performed independently on each of these options based on their respective prices. This is the motivation for this work, namely to present a unified framework for learning to price *all* the derivatives on a particular underlying stock.

The tool we use for acomplishing this task is the Martingale measure, which relates to the *stock dynamics*. If we can predict the stock dynamics in the risk neutral world, then we can price all derivatives on a particular stock. We summarize the advantages of predicting the risk neutral stock dynamics:

(i) Simultaneously prices all derivatives on a stock.
(ii) All derivative data can be used in learning.
(iii) No-arbitrage constraints exist for the risk neutral dynamics.

In contrast, learning to directly price each option suffers from two problems. The first is that more parameters must be learned, one set for each option. The second is that the data on a single option can be used only to learn to predict that particular option's price. On the other hand, only one set of prediction parameters need be learned for predicting the risk neutral dynamics, and all the option prices can be used to learn this single set of parameters – in effect, more data to learn fewer parameters. Also mentioned above are no-arbitrage constraints, which limit the possible risk neutral measures. The no-arbitrage requirement thus provides an economic constraint to regularize the learning in the right direction, further improving the generalization performance of the system.

The underlying theory for the pricing based on the risk neutral dynamics is that the prices can be computed as expected values of cashflows over the risk neutral stock price dynamics. Often the Martingale measure is not unique, and this is where learning comes in. We develop a framework for *learning* the Martingale measure. We assume that the stock dynamics can be represented on a multinomial tree. Binomial trees have often been used to price options [12, 2, 13, 14]. In this work, we present the framework for general multinomial trees, and illustrate with trinomial trees [15], which is more complicated, more flexible and better illustrate the general principles – in the binomial model, there is no learning because the Martingale measure is unique. For background on option pricing and other financial topics, we suggest [15, 13, 14].

The outline of this paper is as follows: first, we give some basics of multiperiod, multinomial trees and option pricing, before presenting the NN-OPT algorithm. We then give some experimental results (trianing and test) on high-frequency paper trading of IBM stock options based on the learned price prediction. Our results indicate that learning Martingale measures together with no-arbitrage regularization constraints performs best.

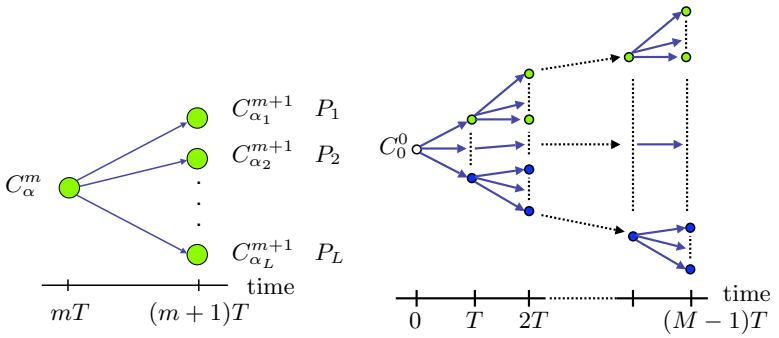
2 Multi-Period Economy with Multinomial Tree

Before introducing NN-OPT, an algorithm to price options, we need set up the notation to describe the economy. Describe the price of an instrument by C

and consider, for example, a 2-period economy (see Fig.1.(a)). Consider time steps m and $m+1$ (corresponding to times mT and $(m+1)T$). At time step m, the instrument could be in one of many states, indexed by α, with price C_α^m. From state α at time step m, assume that the instrument can transition to one of L states, with prices $\{C_{\alpha 1}^{m+1}, \ldots, C_{\alpha L}^{m+1}, \}$. Thus we use $C_{\alpha\beta}^{m+1}$ to denote the possible prices which the instrument can transition into at time step $m+1$ from state α at time step m. When $L=2$, we have a binomial model, $L=3$ is a trinomial model and for $L>3$ a multi-nomial model. Let P_j denote the probability to transition to state j, $j=1, \ldots, L$, and $\sum P_j = 1$. When P_j is independent of m and α, we have a standard multi-nomial tree dynamics for the instrument price. We can represent $C_{\alpha\beta}^{m+1}$ and P_j in vector notation,

$$\boldsymbol{C}_\alpha^{m+1} = \begin{bmatrix} C_{\alpha_1}^{m+1} \\ C_{\alpha_2}^{m+1} \\ \vdots \\ C_{\alpha_L}^{m+1} \end{bmatrix} \quad \text{and} \quad \boldsymbol{P} = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_L \end{bmatrix} .$$

When $m=0$ (time 0), there is only one state C_0^0, and after each time period T, each state can transition to L possible states, which creates a multinomial tree in a multi-period economy (shown in Fig.1.(b)).



(a) 2-period economy (b) M-period economy

Fig. 1. The dynamics of economy

3 Option Pricing

NN-OPT is based on Martingale methods for options pricing and we briefly discuss some background on Martingale pricing. The basic theorem is that the

discounted price is a Martingale with respect to some measure $\boldsymbol{P}$.

$$\begin{aligned} C_{\alpha}^{m} &= D(T) \times E_{\boldsymbol{P}}\left[C_{\alpha\beta}^{m+1}\right] \\ &= D(T) \times \textstyle\sum_{j=1}^{L} P_j C_{\alpha j}^{m+1} \\ &= D(T) \times \boldsymbol{P}^{\boldsymbol{T}} \boldsymbol{C}_{\alpha}^{m+1} \ , \end{aligned} \tag{1}$$

where $D(T)$ is the *risk free discount factor*, which depends on the interest rate and T. Intuitively, this formula means that the current prices are the present value of the expected future prices, where the expectation is with respect to the so called risk neutral probabilities $\boldsymbol{P}$. In this paper, we consider C to be the price of an American option, whose value can be realized by either exercising now or holding and optimally exercising later. Let $G\left(S^m, K\right)$ be the value of exercising at time m with strike K and stock price S^m. Then

$$C_{\alpha}^{m} = \max\left\{G\left(S^m, K\right), D(T) \times \boldsymbol{P}^{T} \boldsymbol{C}_{\alpha}^{m+1}\right\} \ . \tag{2}$$

Thus, we can use backward propagation to compute the current prices of options. To initiate the backward propagation, note that at last time step $(M-1)$, the options don't have any future value, and the option prices become

$$C_{\alpha}^{M-1} = \max\left\{G\left(S^{M-1}, K\right), 0\right\} \ . \tag{3}$$

Therefore, if we know the appropriate values[1] for S^{M-1} (i.e. the stock dynamics), we are able to determine C_{α}^{M-1} for all states at the last time step, and then we use the recursive algorithm to compute the current price C_0^0 of the option. The details of pricing options using multinomial trees can be found in numerous techniques of option pricing, for example [17].

4 The NN-OPT Learning Algorithm

The NN-OPT learning algorithm includes two parts, a standard neural network[2] probability predictor and a multinomial pricing tree. Figure 2 shows the structure of neural networks, where $w_{\eta\delta}^{\theta}$ denote the weights from node η to node δ in layer θ, and the set of weights are denoted by vector $\boldsymbol{w}$. The neural networks will predict and learn the probabilities $\boldsymbol{P}$ for pricing, used in the multinomial pricing tree. The input of the neural networks can be anything, such like short term interest rates, long term interest rates, technical indexes or previous historical data. The output of the neural networks is a set of amplitudes, $\{g_1, g_2, \ldots, g_{L-1}\}$, where $g_i \in [0,1]$. There is also a set of transfer functions $\{H_1, H_2, \ldots, H_L\}$, denoted by a vector $\boldsymbol{H}$, to transfer $\{g_1, g_2, \ldots, g_{L-1}\}$ into probabilities $\boldsymbol{P}$, where,

$$P_i = H_i(g_1, g_2, \ldots, g_{L-1}) \quad i = 1, \ldots, L \ . \tag{4}$$

[1] There are many techniques to determine appropriate values for S^{M-1}, such as historical volatility and GARCH volatility predictors, [12, 16].

[2] The details of the structure and the procedures of a standard neural network, please refer to [18, 19].

The outputs g_i, $i = 1, \ldots, L-1$, are independent of each other. This property makes the process of backward propagation easier (in Sect.4.3). In a trinomial model, one choice for the transfer functions is

$$P_1 = g_1 \tag{5}$$
$$P_2 = (1 - g_1) \times g_2 \tag{6}$$
$$P_3 = 1 - g_1 - (1 - g_1) \times g_2 \ , \tag{7}$$

which satisfy $\sum P_i = 1$ and $0 \leq P_i \leq 1$, for $i = 1, \ldots, 3$.

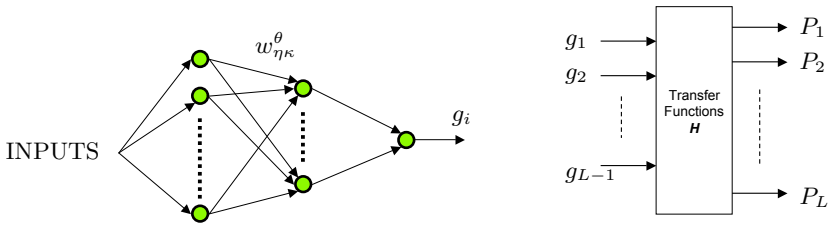


Fig. 2. The structure of the neural network

The second part of the NN-OPT learning algorithm is the multinomial pricing tree which uses the probabilities in $\boldsymbol{P}$ to price all options and compute the feedback to the neural networks for learning (updating of the weights).

4.1 Forward Propagation: Computation of Current Price C_0^0

Figure 3 shows the framework for forward propagation. Assume that we are given a data set, $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$, where $\boldsymbol{x}_i$ is the input vector, and y_i is the expected output. The standard neural networks just apply the regular forward propagation to obtain $\{g_1, g_2, \ldots, g_{L-1}\}$, see for examples [18, 19]. Then using the transfer functions $\boldsymbol{H}$, we calculate pricing probabilities $\boldsymbol{P}$. The pricing algorithm for the multinomial tree is as follows:

1. Use (3) to compute C_α^{M-1} for all states at last time step $(M-1)$.
2. Loop from $i = (M-2)$ to 0,
   – Use (2) and $\boldsymbol{C}_\alpha^{i+1}$ to compute C_α^i for all states at time period i .
3. Output C_0^0.

4.2 Computation of error function $Error(\boldsymbol{w})$

We define the error function as

$$Error(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \left(C_0^0 - y_i\right)^2 \ . \tag{8}$$

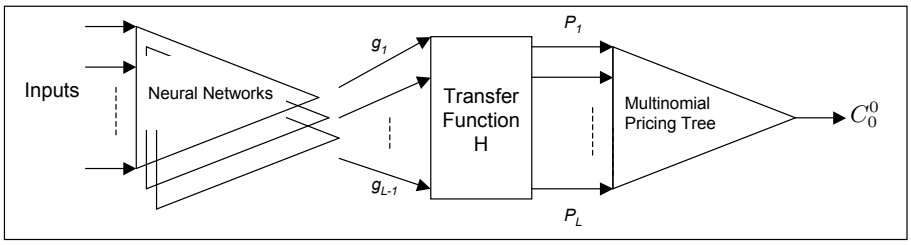


Fig. 3. The framework for forward propagation

The goal is to find a set of weights $\boldsymbol{w}$ that minimizes $Error(\boldsymbol{w})$. To do this, using a gradient descent algorithm, we will need to backpropagate gradients through a multinomial tree and then through the neural network. Backpropagation of gradients through neural networks is standard (see for example [18, 19]). We now develop an efficient algorithm for backpropagating gradients through a multinomial tree pricing framework.

4.3 Backward propagation: Computation of the Gradients

The framework for backward propagation is shown in Fig.4. In order to implement the gradient descent algorithm, we need

$$\frac{\partial Error(\boldsymbol{w})}{\partial w_{\eta\delta}^{\theta}} = \frac{2}{N}\sum_{i=1}^{N}\left(C_0^0 - y_i\right)\frac{\partial C_0^0}{\partial w_{\eta\delta}^{\theta}} \ . \tag{9}$$

There are $L-1$ neural networks, and we need to compute (9) for each one. We will focus on a particular neural network $j \in \{1, 2, \ldots, L-1\}$. In (9), C_0^0 and y_i are known after forward propagation, and we have

$$\frac{\partial C_0^0}{\partial w_{\eta\delta}^{\theta}} = \frac{\partial C_0^0}{\partial g_j} \times \frac{\partial g_j}{\partial w_{\eta\delta}^{\theta}} \ , \tag{10}$$

thus, we need $\partial C_0^0/\partial g_j$ and $\partial g_j/\partial w_{\eta\delta}^{\theta}$ to compute derivatives, (9). For the second term, $\partial g_j/\partial w_{\eta\delta}^{\theta}$, we just apply the regular backward propagation on a standard neural network, so we won't discuss it further. In the following, we will focus on the process of backward propagation in the multinomial pricing tree to determine $\partial C_0^0/\partial g_j$.

From (2) and (3), we know

$$C_{\alpha}^{m} = \begin{cases} \max\left\{G\left(S^m, K\right), 0\right\} & \text{when } m = M-1 \\ \max\left\{G\left(S^m, K\right), D(T)\boldsymbol{P}^T\boldsymbol{C}_{\alpha}^{m+1}\right\} & \text{when } m = 0, \ldots, M-2 \end{cases} , \tag{11}$$

then

$$\frac{\partial C_{\alpha}^{m}}{\partial g_j} = \begin{cases} \partial \max\left\{G\left(S^m, K\right), 0\right\}/\partial g_j & \text{when } m = M-1 \\ \frac{\partial \max\left\{G(S^m,K), D(T)\boldsymbol{P}^T\boldsymbol{C}_{\alpha}^{m+1}\right\}}{\partial g_j} & \text{when } m = 0, \ldots, M-2 \end{cases} . \tag{12}$$

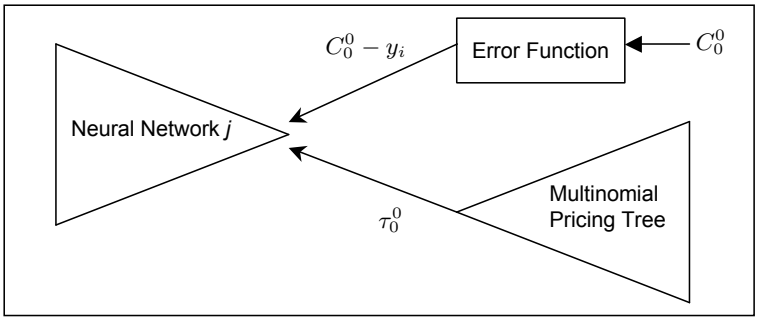


Fig. 4. The framework for backward propagation

Let $\tau_\alpha^m = \partial C_\alpha^m / \partial g_j$,

– Case 1: $C_\alpha^m = G\left(S^m, K\right)$. We know S^m and K are not related to g_j, then

$$\tau_\alpha^m = \frac{\partial G\left(S^m, K\right)}{\partial g_j} = 0 \ . \tag{13}$$

– Case 2: $C_\alpha^m = D(T)\boldsymbol{P}^T \boldsymbol{C}_\alpha^{m+1}$. Then,

$$\begin{aligned}
\tau_\alpha^m &= \frac{\partial \left(D(T)\boldsymbol{P}^T \boldsymbol{C}_\alpha^{m+1}\right)}{\partial g_j} \\
&= D(T) \sum_{i=1}^{L} \frac{\partial P_i C_{\alpha i}^{m+1}}{\partial g_j} \\
&= D(T) \sum_{i=1}^{L} \left(\frac{\partial P_i}{\partial g_j} C_{\alpha i}^{m+1} + P_i \frac{\partial C_{\alpha i}^{m+1}}{\partial g_j} \right) \\
&= D(T) \sum_{i=1}^{L} \left(\frac{\partial P_i}{\partial g_j} C_{\alpha i}^{m+1} + P_i \tau_{\alpha i}^{m+1} \right) \ .
\end{aligned} \tag{14}$$

We have discussed the value of $D(T)$ (in Sect.3), and from (4), we know P_i is the output of $H_i\left(g_1, g_2, \ldots, g_{L-1}\right)$. Then, in (14), we need to determine the rest of three terms, $C_{\alpha i}^{m+1}$, $\partial P_i / \partial g_j$, and $\tau_{\alpha i}^{m+1}$.

Computation of $\boldsymbol{C_{\alpha i}^{m+1}}$. In the forward propagation, Sect.4.1, when calculating C_0^0, we also compute $C_{\alpha i}^t$ for all states at all of the time steps, where $t = 0$, $\ldots$, $(M-1)$, and we just save those values for backward propagation.

Computation of $\boldsymbol{\partial P_i / \partial g_j}$. From (4), we obtain

$$\frac{\partial P_i}{\partial g_j} = \frac{\partial H_i\left(g_1, g_2, \ldots, g_{L-1}\right)}{\partial g_j} \ . \tag{15}$$

In the beginning of Sect.4, we discuss $\boldsymbol{H}$ which are known because we construct the neural networks and decide those transfer functions, and we also know g_j is independent of $\{g_1, \ldots, g_{j-1}, g_{j+1}, \ldots, g_{L-1}\}$. Then, (15) is solvable. From the same trinomial example, see (5)(6), and (7), and then (15) can be computed easily; for instance,

$$\frac{\partial P_3}{\partial g_1} = \frac{1-g_1-(1-g_1)\times g_2}{\partial g_1} \quad .$$

Computation of $\boldsymbol{\tau_{\alpha i}^{m+1}}$. From (14), we know τ_α^m at time step m can be computed from $\tau_{\alpha j}^{m+1}$, $j = 1, \ldots, L$, at next time step $m+1$, and from (12) and (13), we know τ_α^{M-1} for any state at last time step are 0. Then, we can use backward propagation for the multinomial tree to compute τ_α^m for any state at any time step. The algorithm is as follows:

1. Set all τ_α^{M-1} to 0.
2. Loop from $i = (M-2)$ to 0,
   - Use (14) to compute τ_α^i for all states at time step i from the value of $\tau_{\alpha_j}^{i+1}$, $j = 1, \ldots, L$.
3. Output τ_0^0.

Now, we have determine the derivative of error function $Error(\boldsymbol{w})$, (9) and then apply the regular process to update the set of weights $\boldsymbol{w}$ in standard neural networks. Then repeat the all process for each iteration of learning.

5 Results

We developed a simple trading system to evaluate the NN-OPT, which we tested using intraday real market data (5 minutes time period) for IBM (stock and option data) and interest rate data, from July 20, 2004 to April 29, 2005. We used the first 80 days, from July 20, 2004 to November 9, 2004, as the training data set, and used the remaining 118 days as test data set (see Fig.5). We compared the trading performance between different algorithms and different trading costs. Some algorithms, using NN-OPT with no-arbitrage constraint to learn *risk-neutral probabilities*, [2, 6]. Intuitively, arbitrage is the possibility to make money out of nothing.

1. **Enforcing No-arbitrage, with learning**: This is based on a no-arbitrage condition and also uses NN-OPT learning algorithm to predict the *risk-neutral probabilities* for pricing.
2. **Not Enforcing No-arbitrage, with learning**: This approach is only based on NN-OPT learning algorithm without no-arbitrage constraints.
3. **Enforcing No-arbitrage, no learning**: This approach is to demonstrate that a no-arbitrage constraint alone, without learning the Martingale measures is worse than our framework.
4. **Not Enforcing No-arbitrage, no learning (random strategy)**: This approach is to develop a benchmark performance using a random strategy.

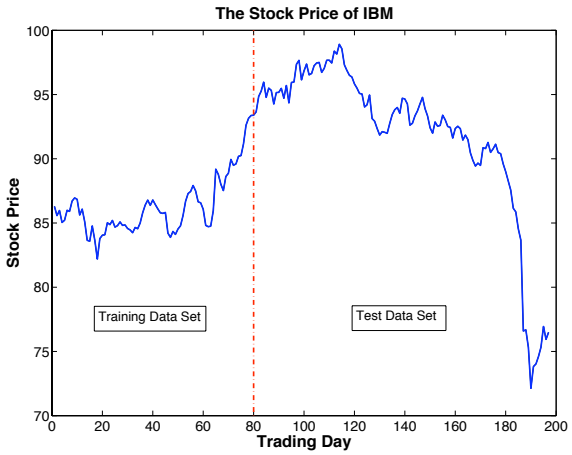


Fig. 5. The intraday market price of IBM

The results of trading using these approaches are shown in Fig.6, and the figure on right hand side has higher trading cost. Using both no-arbitrage constraint and NN-OPT clearly has the best performance. Note that the system still makes money even when the market crashes. As we move further from the training window, the performance degrades, though it remains positive. The results of the other algorithms are also reasonable because any random trading strategy will systematically lose the transaction cost on each trade which means that the total profit will drop linearly; the results also show that it is useful to use a no-arbitrage condition because it narrows the range of Martingale measures to obtain a set of plausible prices, rather than pure random.

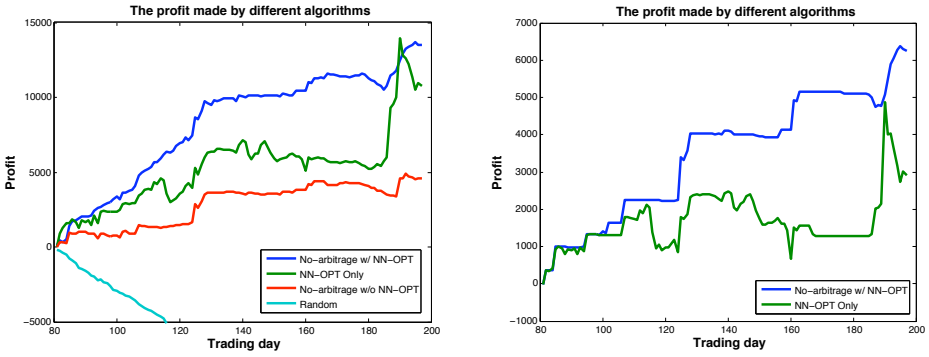


Fig. 6. Comparison of the trading results of stock IBM using different algorithms

References

1. Black, F., Scholes, M.S.: The pricing of options and corporate liabilities. Journal of Political Economy **3** (1973) 637–654
2. Magdon-Ismail, M.: The Equivalent Martingale Measure: An Introduction to Pricing Using Expectations. IEEE Transactions on Neural Netork **12**(4) (2001) 684–693
3. Cox, J.C., Ross, S.A.: The valuation of options for alternative stochastic processes. Journal of Financial Economics (1976) 145–166
4. Back, K., Pliska, S.R.: On the fundamental theorem of asset pricing with an infinite state space. Journal of Mathematical Economics (1991) 1–18
5. Harrison, J.M., Kreps, D.: Martingales and arbitrage in multiperiod securities markets. J. Economic Theory **20** (1979) 381–408
6. Harrison, J.M., Pliska, S.R.: Martingales and stochastic integrals in the theory of continuous trading. Stochastic Processes and their Applications **11** (1981) 215–260
7. Harrison, J.M., Pliska, S.R.: A stochastic calculus model of continuous trading: Complete markets. Stochastic Processes and their Applications (1983) 313–316
8. Musiela, M., Rutkowski, M.: Martingale Methods in Financial Modeling (Applications of Mathematics, 36). New York: Springer-Verlag (1977)
9. Paul R. Lajbcygier, J.T.C.: Improve option pricing using artificial neural networks and bootstrap methods. International Journal of Neural System **8**(4) (1997) 457–471
10. Amari SI, Xu L, C.L.: Option pricing with neural networks. In Progress in Neural Information Processing **2** (1996) 760–765
11. Moody, J. Saffell, M.: Learning to trade via direct reinforcement. IEEE Transactions on Neural Networks **12**(4) (2001) 875–889
12. Cox, J.C., Ross, S.A., Rubinstein, M.: Option pricing: A simplified approach. Journal of Financial Economics (1979) 229–263
13. Ross, S.M.: An Elementary Introduction to Mathematical Finance. Second edn. Cambridge University Press (2003)
14. Wilmott, P., Howison, S., Dewynne, J.: The Mathematics of Financial Derivatives. Cambridge University Press (1995)
15. Baxter, M., Prnnie, A.: Financial Calculus: An Introduction to Derivative Pricing. Cambridge University Press (1996)
16. Bollerslev, T.: Generalized autoregressive conditional heteroskedasticity. Journal of Econometrics **31** (1986) 307–327
17. Kargin, V.: Lattice option pricing by multidimensional interpolation. Mathematical Finance **15**(4) (2005) 635–647
18. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press (1995)
19. Haykin, S.: Neural Networks: A Comprehensive Foundation. 2nd edn. Prentice Hall (1998)