Machine learning tools for option pricing

The non-parametric modelling alternative

Ed Harvey

June 11, 2023

1 Introduction

in the 50 years since the publication of the Black-Scholes-Merton model (BSM) in Black & Scholes (1973) and Merton (1973), there has been a rapid increase in the use of financial derivatives with non-linear payoffs (such as the simple option shown in Figure 1). The model provides an analytical formula for pricing European-style options (meaning exercisable only at a given maturity) by employing stringent assumptions about the expected future path of the price of the underlying asset (geometric brownian motion (GBM) with constant volatility), and the nature of the market for both the underlying and cash instruments (frictionless and efficient). The inputs required by the model are details of the option (strike price and time remaining to maturity), the current market (price of the underlying), and expected future conditions (risk-free interest rate and volatility of the price of the underlying).

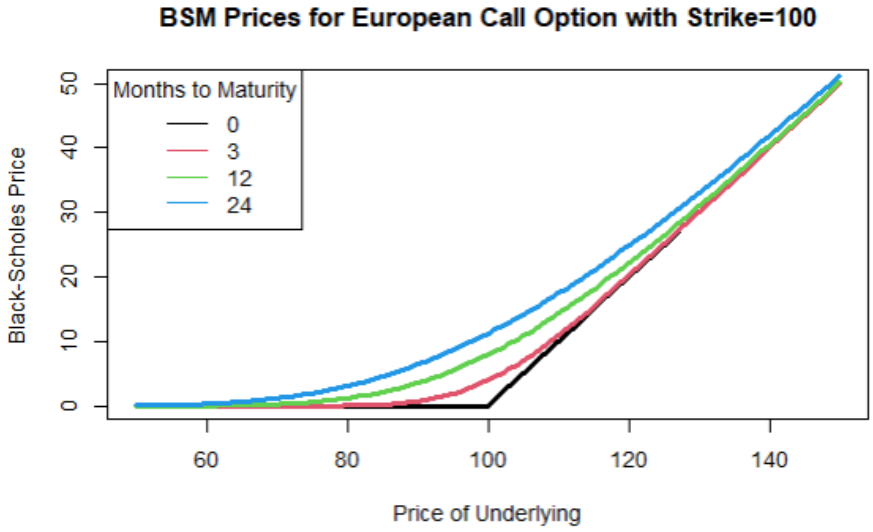

Figure 1: Chart showing BSM Options valuations with decreasing times to maturity - the black line is the final payoff at maturity

The BSM is a parametric model requiring that the parameters for its simple model of market dynamics are estimated from past performance or inferred from market prices. Its main advantage, and reason for continued use in financial markets to this day, is speed and simplicity of use for traders. However, any parametric model for pricing has the disadvantage of requiring assumptions about the distribution of future changes in market prices. In the case of the BSM, these assumptions are particularly strict and are often observed to be broken in practice. For example:

- Stoyanov et al. (2011) explain that empirical distributions often exhibit fat tails, where equity prices mostly move very little on a daily basis and then, occasionally but more often than a normal distribution of returns would suggest, by a great deal. The distributions also often exhibit skew. In the case of equities, this often means that large falls are more likely than similarly sized gains.

- Derman & Kani (1994) show that options are traded in financial markets at prices which would imply different volatilities depending on the exercise price, showing that some of the participants are well aware of the non-normality of the distributions of returns.

In particular, the fat tails mean that BSM under-estimates the likelihood of large price moves in the underlying asset. In times of high volatility in the market, as in market crashes such as the credit crisis of 2007-2009, this can mean that sellers of options are not fully aware of their vulnerability to large losses.

Consequently, there has long been interest in improving on the BSM. Non-parametric approaches are one option, where the idea is to use machine learning techniques to train a model (ML model), using historical data, to price an option given current market data only. These approaches can avoid relying on restrictive assumptions and, while requiring significant training time, can be very fast to calculate a price for a previously unseen option once trained.

For example, Hutchinson et al. (1994) replicated the prices of the BSM using ML models trained on time-series of underlying asset prices - firstly, simulated assuming geometric brownian motion and, secondly, taken from historical real-world prices. They then tested these in two ways - by calculating the correlation between prices calculated by each ML model and the BSM, and by comparing the tracking errors of using each ML model and the BSM to hedge a set of options. I propose to look at these ML models again and experiment with modern machine-learning algorithms included within standard packages. I also propose to review an extension to the ML models to price more complex options.

2 Research Objectives

1. Re-perform the main experiment from Hutchinson et al. (1994) showing that the non-linear function of a BSM option price can be accurately replicated using ML models based on multi-layer perceptrons.
2. Code this using standard packages available in R.
3. Experiment with multiple setups to obtain the best performance in terms of matching the price and reducing tracking errors while minimising the time spent training the ML model.
4. Create functions to allow a user to train an ML model, or load up a previously-trained one, in order to quickly obtain a price for a new option as well as its 'Greeks' (the sensitivity of the option price to changes in the inputs).
5. Explore the extensions described in Section 5.

3 Experimental Methods

3.1 Models

Hutchinson et al. (1994) reviewed three different learning network models (Radial Basis Functions, Multi-Layer Perceptrons and Projection Pursuit Regression) and did not find huge differences in the performance of these techniques. I will use Multi-Layer Perceptrons (MLPs) for my experiments as these have been adopted widely over the last 30 years since the paper and there are many open-source software packages available to aid in implementing them.

3.1.1 Single Perceptron

In order to understand an MLP, it is first necessary to understand a single perceptron unit (see Figure 2). This is a simplified artificial version of the neurons in a human brain. It was first proposed by McCulloch & Pitts (1943) and the first implementation was reported in Rosenblatt (1958). Given a vector of inputs $(1, x_1, ..., x_n)$, it applies weights $(w_0, w_1, ..., w_n)$ to each input (w_0 is known as the bias), sums the results, and puts the result through a non-linear activation function g to supply an output $\hat{y} = g(\underline{x} \cdot \underline{w} + b)$.

There are a number of commonly-used choices for $g(z)$, for example, $\sigma(z) = \frac{1}{1+e^{-z)}}$, gives a result between 0 and 1, $\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$ gives a result between -1 and 1, and $\text{ReLU}(z) = \max(z, 0)$ gives a linearly increasing output for positive values of the input only.

Given a training dataset of m vectors of inputs together with m actual outputs $(y_1, ..., y_m)$, the model weights are adjusted from their random initial setting to minimise some loss function comparing $(y_1, ..., y_m)$

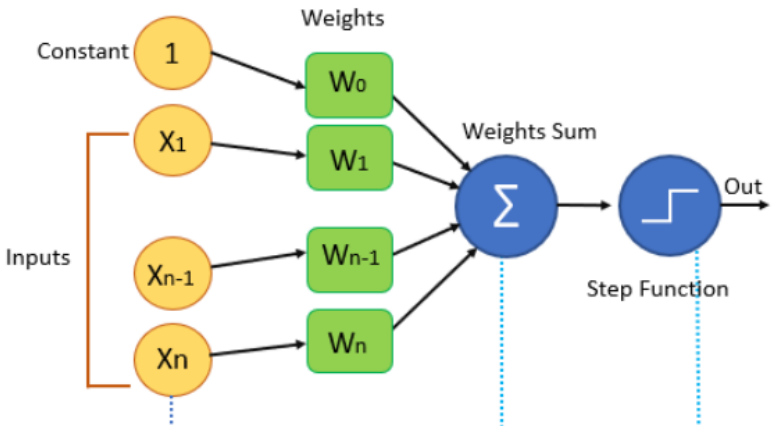

Figure 2: Single Perceptron (Pedamkar 2023)

with $(\hat{y}_1, ..., \hat{y}_m)$. Common loss functions are Mean Squared Error, MSE $= \sum_{j=1}^{m}(y_j - \hat{y}_j)^2$ or Mean Absolute Error, MAE $= \sum_{j=1}^{m} |y_j - \hat{y}_j|$.

An issue with allowing a model to strictly minimise the loss for the training set is that it can over-fit, meaning that it matches the training data very well, including any noise, but does not generalise well to new input data. A regulariser term in the loss function can reduce this effect by adding a penalty such as $\lambda \sum_{i=1}^{n} |w_i|$ to the loss function so that the model trains to reduce the size of the weights as well as the main loss function.

In this instance, the inputs must be normalised to be on a similar scale. Without this, the single λ would affect the weights for some inputs very differently from others.

3.1.2 Loss Function Minimisation

Gradient Descent (GD) is the underlying concept behind the algorithms which aim to minimise the loss function and was first studied for this purpose in Curry (1944). At each stage, each weight w_i is adjusted by the negative of the partial derivative of the output with respect to that weight, multiplied by a learning rate η, resulting in $w_i \to w_i - \eta \frac{\partial \hat{y}}{\partial w_i}$. This continues until some convergence criterion is met (e.g. change in loss function is below a certain level).

One issue is that choosing the learning rate is key. Too large, and it will oscillate around the minimum value it is aiming for and never meet it. Too small, and it will require a very long training process to find it. Momentum adjustments memorise the previous steps to allow the algorithm to increase the learning rate automatically when moving consistently in one direction. RMSprop, from Hinton (2012), and AdamW, from Loshchilov & Hutter (2019), are commonly-used optimisers which employ different methods of averaging previous steps to amend this basic momentum idea.

A second issue is that the loss function and its derivatives may take a long time to calculate. This issue is reduced by updating the weights after each 'batch' of training data rather than for the whole dataset. However, Stochastic Gradient Descent (SGD) is used to further improve training speed, as described in LeCun et al. (2012), by using a single, randomly-chosen training data instance to calculate the derivative. This increases the variance of the adjustments to the weights at each step of the SGD process but speeds up the calculations so that training time can be reduced.

A third issue is that there may be multiple local minima corresponding to different sets of weights (see Figure 3). It is then difficult for the algorithm to find the global minimum. A useful effect of the increased variance in the adjustments to the weights resulting from SGD, is that the optimisation may be able to break out of a search for a local minimum and find other local minima.

However, finding the global minimum can still depend heavily on the random initial settings for the weights. For this reason, ensembles of models can be built, each trained with a different set of random initial weights. The average of the results of these models can be used when calculating a new price or, for speed, the model with the lowest value of the loss function.

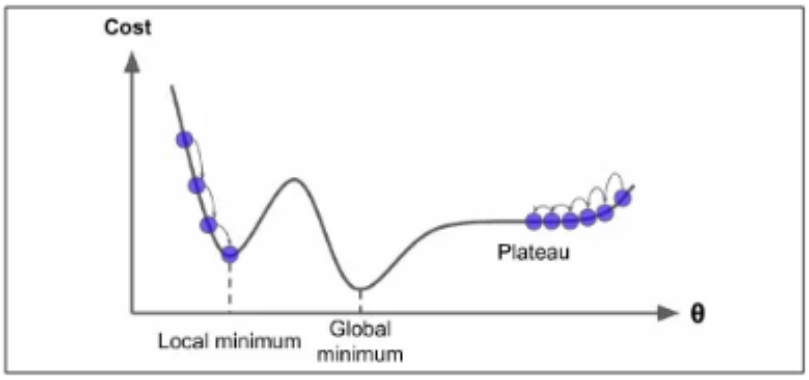

Figure 3: Local Minimum Problem (Yashwanth 2020)

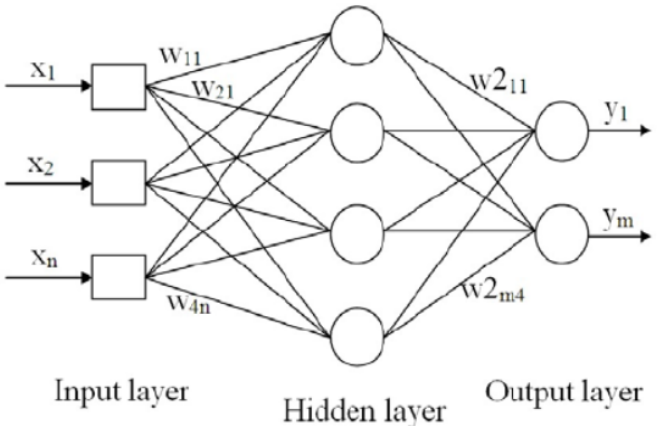

Figure 4: Diagram of a Multi-Layer Perceptron (Mokhtar & Mohamad-Saleh 2013)

3.1.3 Multi-Layer Perceptron

A single perceptron (or single layer of perceptrons) can only train on linearly separable datasets. If the function being matched is more complex, an MLP is likely to improve performance.

An MLP has hidden layers of multiple perceptrons - see Figure 4 for a 2-layer example with 1 hidden layer. Every input in a data point in the input layer feeds every unit in the first layer (meaning it is fully-connected), each with a different set of weights and biases. The outputs of that layer feed the next layer, again with a full set of different weights and biases, and so on until the outputs of the final hidden layer feed an output layer. For this project, there is only one output in the output layer, representing the ML model calculation of the value of the option.

The loss minimisation in the training process requires back-propagation of adjustments from the outputs through each of the hidden layers in turn as shown experimentally in Rumelhart et al. (1986), but the issues discussed in Section 3.1.2 still apply.

3.1.4 Hyper-Parameter Optimisation

There are a large number of choices that can be made when implementing an ML model such as MLP. These are often referred to as hyper-parameters to distinguish them from the weights being trained to allow the model to predict outputs later. For example:

- The number of units in each hidden layer (the number of units in the input and output layers are determined by the problem).
- The number of hidden layers.
- The loss function.
- The activation function for each layer.
- The optimisation algorithm to employ in minimising the loss function.
- Certain inputs to the optimisation algorithms that adjust how they operate.

- The regulariser and its size.
- Number of epochs - the number of times to loop through all the training data during the training process.
- Batch size - the number of sets of training inputs to test before updating the weights.
- Number of models to include in an ensemble.

I will experiment with a range of possibilities by testing them on a validation set of data (to be created in the same way as the final testing data as described in Section 3.2). In many cases, the default settings in the Keras package give a useful base value for the hyper-parameters. In others, I will start simple (e.g. one hidden layer) then increase model complexity until I reach a point where no further performance improvement is obtained.

3.1.5 R Implementation

Open-source packages have been built to enable the Python implementations of TensorFlow and Keras to be used in the R computing language. These allow a range of models to be built with a chosen number of layers and units, an activation function and regulariser to use for each layer, and a loss function and optimiser defined. These models can then be trained, when the number of epochs and the batch size are defined. Finally, the model can be used to predict the correct output from a set of input data that was not used during training.

The code example below would build a standard MLP with 1 hidden layer of 200 units, a ReLU activation function, a regulariser, an MSE loss function and an RMSProp optimiser. It would train its training inputs and output using 5 epochs and a batch size of 128, and would then produce a predicted set of outputs from testing inputs:

```
model = function(ncols) {models = keras_model_sequential() %>%
layer_dense(units = 200, activation = "relu", input_shape = ncol(training_inputs),
    regularizer_l2(0.01)) %>%
layer_dense(units = 1) %>%
compile(optimizer = "rmsprop", loss = "mse", metrics = c("mae"))}

learns = model %>%
fit(x = training_inputs, y = training_output, epochs = 5, batch_size = 128,
    validation_split = .2, verbose = TRUE,)

model %>% predict(testing_inputs)
```

3.2 Data

For the main research objectives, the training data will be:

Inputs:

- Simulations of daily underlying asset prices over a 2 year time period. These are based on the assumption that the distribution is described by GBM - that is, for an initial price $S(0)$ at time $t = 0$, the price $S(t)$ at time $t > 0$ is determined by the formula $S(t) = S(0)e^{\sum_{i=1}^{n} Z_i}$, where n is the number of incremental time periods δt from 0 to t, and Z_i is a draw from the standard normal distribution $N(\mu\delta t, \sigma^2\delta t)$, where $\mu = 0.05$ is the annual drift and $\sigma = 0.2$ the annual standard deviation of the value being simulated.
- A range of maturity dates for reference European call options. These are for each month-end date greater than the current simulation date up to half a year beyond the final simulation.
- A random range of exercise prices for reference European call options. These are based on a uniform distribution between 0.5 and 2 times the current underlying asset price. Given the σ and maximum time to maturity, the function for the option price outside these values becomes close to linear.

Output:

- Daily prices for each European call option described in the inputs. These are based on the analytical solution included in the BSM - that is, the value $C(t)$ is calculated using $C(t) = S(t)N(d_1) - Xe^{-r(T-t)}N(d_2)$ where X is the strike price of the option, r is the risk-free interest rate, T is the maturity data of the option and $d_1 = \frac{ln\left(\frac{S(t)}{X} + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right)}{\sigma\sqrt{T-t}}, d_2 = d_1 - \sigma\sqrt{T-t}$.

To allow the ML model to be tested on a set of call options which has not been used in the training process, a similar simulation of underlying asset prices plus reference call options will create testing inputs and the ML model will be used to calculate values of the option for each of these.

Each of the training inputs will be normalised by subtracting its mean and dividing by its standard deviation. The testing inputs will be normalised using the mean and standard deviations of the training inputs in order to treat new data in exactly the same way as the data that the model was trained on.

3.3 Tracking Errors

The BSM assumes 'frictionless' markets which, among other things, means that price moves are continuous and that trading in the underlying asset can be performed continuously and without cost. The possibility then exists for continuous re-balancing of a position in the underlying asset to hedge any risk that the outstanding option position changes in value. The amount of the asset required to do this, for an option on a single unit of the underlying, is given by the derivative (or 'sensitivity') of the option price with respect to the price of the underlying (in financial markets jargon, this is the 'delta', one of the 'Greeks'). If this process is performed continuously throughout the life of the option, then the total of cash and value of underlying immediately after the payoff from the option will equal zero.

In the real world, this re-balancing of the hedge can only be performed in discrete steps and also incurs trading costs. I will not model the effect of trading costs, but I will use the simulated underlying asset prices to re-balance the hedge discretely on a daily basis. The result is a path of valuations of the trading strategy as a whole (cash + value of option + value of underlying asset position). The absolute value of this at maturity of the option is then the 'tracking error' of the hedging strategy.

3.4 Evaluation

The models will be compared to each other and to the results in Hutchinson et al. (1994) using the following measures:

1. The R-squared measure on the series of ML model and BSM call option prices. The closer this is to one, the more correlated the ML model prices are to the BSM ones.
2. The mean of the difference between the ML model and BSM prices. A figure nearer to zero means that there is little bias in the difference.
3. The tracking errors using the ML model and BSM, as described in Section 3.3.
4. The time taken to train the ML model. The modern ML models will be faster than those from 1994 due to increased processing power but the comparison to each other will be part of the decision on what is most useful in practice.
5. The time taken to calculate the price of a set of previously unseen options.

4 Practical Implications

Although non-parametric models have not been heavily used in pricing and hedging in financial markets to date, they have advantages that could be used in several practical situations.

- Day-to-day trading and valuation of portfolios of options. This project is a proof-of-concept by matching prices for a simple non-linear financial instrument using a simple model of underlying asset price movements. It can show that creating an ML model to value these is feasible, and therefore could also be possible for more complex situations which would otherwise require time-consuming methods to solve, meaning a long window before a price can be given to a customer or a full valuation of a portfolio can allow a trader to see the risks that need to be hedged. For example:

- A more complex option (e.g. hybrids such as convertible bonds or basket options where the option is exercisable on more than one underlying asset).
- A more complex model of the underlying, e.g. Bouchard-Potters as discussed in Raberto (1999), which may better model the fat tails and skew of real-world distributions.

Pricing these using ML models could exchange the lengthy periods required to train the ML model on multiple examples of the results of the model described above, against the much shorter periods required to obtain a price when it is more urgent for trading purposes.

- Finding market mis-pricing. Time series of historical underlying prices could be used to value a range of options (especially across exercise prices) to find where the relationships between those market prices appear inconsistent with the historical evidence of dynamics of the underlying asset's price movements. This could provide trading opportunities or better risk control, depending on the outlook of the market participant.

5 Extensions

I will perform a similar experiment to that described in the earlier sections, but for an option which would require some form of numerical analysis to value in the BSM world. This will help to illustrate the point made in Section 4 around reducing the time taken to obtain a price once the time-consuming training process has been completed.

I will also provide a user interface to allow a user to input details of options or models to be used and obtain prices and sensitivities.

6 Project Timeline

Name	Begin date	End date
Write Proposal	31/05/2023	12/06/2023
ML Model matching BSM	22/05/2023	30/06/2023
Testing for one model	22/05/2023	15/06/2023
Testing for best of multiple models	19/06/2023	23/06/2023
Create best model and user functions	26/06/2023	30/06/2023
ML Model matching more complex model	03/07/2023	21/07/2023
Testing for one model	03/07/2023	07/07/2023
Testing for best of multiple models	10/07/2023	14/07/2023
Create best model and user functions	17/07/2023	21/07/2023
Create User Interface	24/07/2023	11/08/2023
Research Shiny app creation	24/07/2023	28/07/2023
Create Shiny app with model and valuation functions	31/07/2023	11/08/2023
Write up dissertation	20/06/2023	18/08/2023
Introduction	20/06/2023	17/07/2023
Related work	26/06/2023	21/07/2023
Methods	03/07/2023	28/07/2023
Discussion	31/07/2023	11/08/2023
Conclusion	10/08/2023	16/08/2023
Abstract	14/08/2023	16/08/2023
References	20/06/2023	16/08/2023
Final check for submission	16/08/2023	18/08/2023

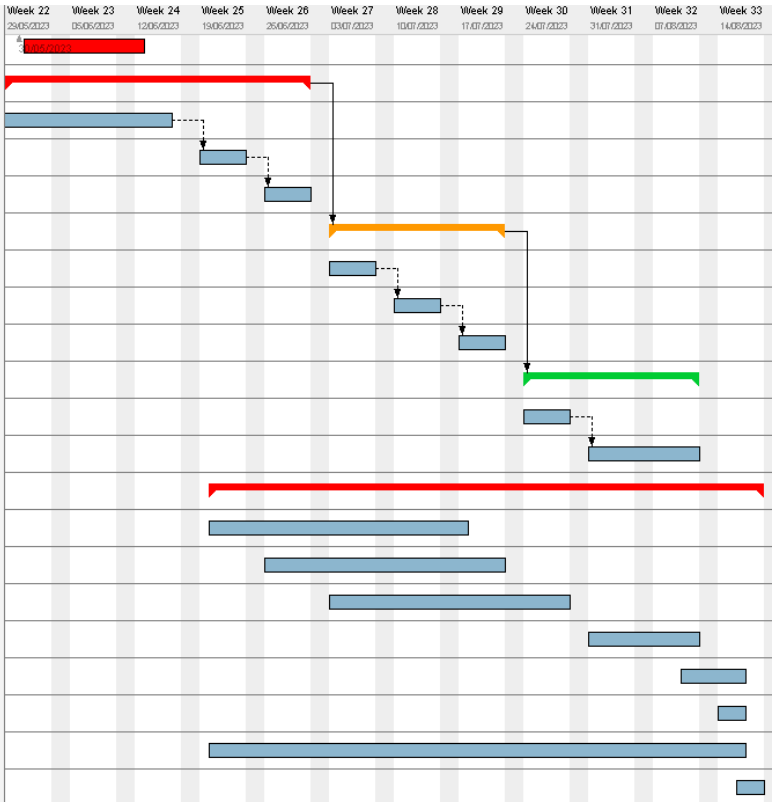

Figure 5: Gantt representation of the project timeline

References

Black, F. & Scholes, M. (1973), 'The pricing of options and corporate liabilities', *Journal of political economy* **81**(3), 637.

Curry, H. B. (1944), 'The method of steepest descent for non-linear minimization problems', *Quarterly of Applied Mathematics* **2**, 258–261.
URL: *https://www.ams.org/journals/qam/1944-02-03/S0033-569X-1944-10667-3/S0033-569X-1944-10667-3.pdf*

Derman, E. & Kani, I. (1994), 'Riding on a smile', *Risk* **7**.

Hinton, G. (2012), 'Lecture 6a overview of mini-batch gradient descent'.
URL: *http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture$_s$lides$_l$ec6.pdf*

Hutchinson, J., Lo, A. & Poggio, T. (1994), *A nonparametric approach to pricing and hedging derivative securities via learning networks*, number 4718 *in* 'NBER working paper series', National Bureau of Economic Research, Cambridge, Mass.

LeCun, Y., Bottou, L., Orr, G. B., & Muller, K.-R. (2012), *Neural networks: Tricks of the trade*, Springer Berlin Heidelberg.

Loshchilov, I. & Hutter, F. (2019), 'Decoupled weight decay regularization'.

McCulloch, W. S. & Pitts, W. (1943), 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**, 115–133.

Merton, R. C. (1973), 'Theory of rational option pricing', *The Bell Journal of economics and management science* pp. 141–183.

Mokhtar, K. Z. & Mohamad-Saleh, J. (2013), 'An oil fraction neural sensor developed using electrical capacitance tomography sensor data', *Sensors (Basel, Switzerland)* **13**, 11385–406.

Pedamkar, P. (2023), 'Perceptron learning algorithm'.
URL: *https://www.educba.com/perceptron-learning-algorithm/*

Raberto, M. (1999), Processi stocastici e reti neurali nell-analisi di serie temporali finanziarie, PhD thesis, Università di Genova, Facoltà di scienze matematiche, fisiche e naturali.

Rosenblatt, F. (1958), 'The perceptron: a probabilistic model for information storage and organization in the brain', *Psychology Review* **65(6)**, 386–408.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986), 'Learning internal representations by error propagation', *Parallel distributed processing: Explorations in the microstructure of cognition* **1**.
URL: *https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf*

Stoyanov, S. V., Rachev, S. T., Racheva-Iotova, B. & Fabozzi, F. J. (2011), 'Fat-tailed models for risk estimation', *Working Paper Series in Economics* **30**.
URL: *https://www.econstor.eu/obitstream/10419/45631/1/659400324.pdf*

Yashwanth, N. (2020), 'Understanding gradient descent'.
URL: *https://medium.com/analytics-vidhya/gradient-descent-and-beyond-ef5cbcc4d83e*