Computing implied volatility via inverse neural networks

Le Phuong Quynh

MSc Thesis

Consultant: Dr. András Zempléni

Supervisor: Kinga Tikosi

Department of Probability Theory and Statistics

Faculty of Science

Eötvös Loránd University

Budapest, 2021

Contents

List of Figures

List of Tables

1 Introduction

Volatility is one of the most critical market parameters that governs the price evolution of financial instruments and derivatives. For decades, scientists and practitioners have grappled with the issue of determining implied volatility. There have been several attempts to develop a method or algorithm to calculate uncertainty from option prices.

The main approach for computing implied volatility is to find an explicit or implicit expression for the volatility which can be derived from the financial option pricing models. Black - Scholes model for option pricing is one of the most popular financial model that can be used to extract the implied volatility from European option price formula. Coming from the nature of Black - Scholes formula, a closed form of implied volatility can not be found [27] but can only be approached by approximation techniques. A huge number of work have been developed in seeking for a closed formula approximation [9], [8], [10] in the 1990s and this effort did not be stamped out even in early 2000s [11]. Raised by Donald in 2001 [12], an extension has been applied for the Chance model (1996) to get a formula that was simpler and much more accurate. In this article, the author also pointed out the limitation in existing calculation formula and its low percentage of accuracy. All kind of methods in reality is limited to only some specified cases, in other conditions they diverge far from the true values.

A more practical approach is to approximate by numerical iteration methods, which was mentioned early in 1982 [13]. Iteration procedure might meet difficulties with inappropriate starting point, dividing by zero derivatives or disability of finding root if facing local maxima, minima. Several modified extensions and improvements have been made to minimizes the disadvantage cases. These methods may meet the requirement of approximated accuracy. In [14], authors summarized and commented about some different approaches in determining the implied volatility both approximation methods and numerical approaches.

Nevertheless, the interest for a better method does not seem to decline. With the development of deep learning, especially deep neural networks [34], neural network models were built for forecasting the implied volatility. Neural networks can forecast with low error the realistic trading volatility [54] together with ability to canceling drawbacks of iteration process. There is a lot of literature research and tests on forecasting volatility with the usage of artificial neural networks and the hybrid of neural network with other models, example of works raised early in 1990s [54], [57] and more recently [53], [55], [56], [58].

The aim of this study is to propose a literature implementation of a new approach extending from forward training neural networks. We further investigate the feasibility of this approach by testing the work process of two simple models with a random gener-

ated data set. The accuracy of this new approach is assessed by putting in comparison with an original forward neural network together with the Newton - Raphson iteration method.

The rest of this thesis is organized as follows. In the next section preliminaries about financial terms and mathematical notation including Brownian motion, Ito process are introduced for the sake of further understanding on financial models' construction. The third section and the fourth section are used for understanding and constructing famous financial option pricing models, the Bachelier model and the famous Black - Scholes model, and volatility emphasized on implied volatility. Section 5 is devoted neural networks and the work of me on testing and choosing appropriate hyper-parameters before the final network construction. The algorithm of the new approach is presented in section 6. The rest is for numerical results and conclusions.

2 Preliminaries

2.1 What is an option?

The most financial term that is close to an option is a contract. One common example is the forward contract. The two parties of a forward contract agree to buy or sell an asset for a fixed price at a certain time in the future. An option, like a forward contract, allows the parties to buy or sell an amount of stocks with a predetermined price by a certain date. A call option gives the right to buy while a put option allows its owner to sell the underlying assets. However, different from a forward contract, an option's holder is not obligated to buy or sell the assets.

An option or a contract are different types of terminology derivatives. A derivative can be defined as a financial instrument or a second security whose value is derived from the values of underlying variables. Like a stock option is a derivative whose value depends on the price of the stock. Common underlying products can be stocks, bonds, commodities, currencies, interest rates.

The price in the contract is known as the strike price, the date in the contract is the maturity. Options can be applied to not only stocks but also other various kinds of financial instruments like commodities, bonds, foreign currencies. Based on different criteria, there are several different option styles. American options can be exercised at any time up to the expiration date. European options can be exercised only on the expiration date itself. These are the two most common styles and also called vanilla options. Furthermore, an option can have more complex features and can be customized based on the needs of the investors, which is called an exotic option. It can differ from a vanilla option in its expiration date, strike prices or payment structures. One of the most common types of exotic options is barrier options. A barrier option is similar to a vanilla call or put option but they only become activated or extinguished if the underlying asset price reaches a barrier level (knock-out or knock-in price). In this paper we will only focus on the European call option.

Following are definitions about some concepts that will be used in the whole of this work.

Definition. A universe is a class that includes all of the entities that one would like to consider in a given case.

With the concept of universe, we will assume from here that in our universe there always exists the **risk-free interest rate**, r. This rate also is referred as the only cost of holding a stock over a period of time.

Definition. From the assumption of universe and risk-free rate, we derive the **forward price of a stock** which is the current price of a stock S_0 and an its expected

return after time t of holding it. Then the forward price can be computed as

$$S_0 e^{rt},$$

where r is risk-free rate.

Definition. We define a **risk-neutral universe** when for every asset A and period of time t, the value of asset $C(0, A)$ at $t = 0$ is the expected value of the asset at time t discounted to its present value

$$C(0, A) = e^{-rt}\mathbb{E}[C(t, A)].$$

Before jumping to an important lemma about the mean and variance of log-normally distributed stock price S_t, we denote by σ the annual volatility in the percent rise in the stock price, that is the standard deviation of the percentage change in the price over one year.

Lemma 1. *Let S_0 and S_t be respectively the stock price at initial and at time t. Assume S_t is log-normally distributed, i.e $\ln \frac{S_t}{S_0}$ is normally distributed with mean μ and variance ν and the mean of log-normal distribution is located at the forward price of the stock. Then, with $\mu = \mu(t), \nu = \nu(t)$:*

$$\nu = \sigma^2 t, \tag{1}$$

$$\mu = \left(r - \frac{\sigma^2}{2}\right) t. \tag{2}$$

Proof. It is clear that by the definition of σ, $\ln \frac{S_t}{S_0}$ has variance σ^2 after 1 year. After $t-1$ years, $\ln \frac{S_t}{S_0}$ has variance $\sigma^2(t-1)$ and after t years

$$\begin{aligned}\ln \frac{S_t}{S_0} &= \ln \frac{S_{t-1} S_t}{S_0 S_{t-1}} \\ &= \ln \frac{S_{t-1}}{S_0} + \ln \frac{S_t}{S_{t-1}},\end{aligned}$$

then has variance $\sigma^2(t-1) + \sigma^2 = \sigma^2 t$.

To derive result 2 we consider

$$\begin{aligned}F(a) &= \mathbb{P}(S_t \leq a) \\ &= \mathbb{P}(S_0 e^{x_t} \leq a) \\ &= \mathbb{P}(x_t \leq \ln \frac{a}{S_0}) \\ &= \frac{1}{\sqrt{2\nu\pi}} \int_{-\infty}^{\ln \frac{a}{S_0}} e^{\frac{-(x_t-\mu)^2}{2\nu}} \mathrm{d}x.\end{aligned}$$

Differentiate both sides with respect to a, getting

$$f(x) = \frac{1}{\sqrt{2\nu\pi}x} e^{\frac{-(\ln\frac{x}{S_0}-\mu)^2}{2\nu}},$$

$f(x)$ is the density function of S_t. By the expected of forward price $\mathbb{E}[S_t] = S_0 e^{rt}$,

$$\begin{aligned}\mathbb{E}[S_t] &= \int_0^\infty \frac{1}{\sqrt{2\nu\pi}x} x e^{\frac{-(\ln\frac{x}{S_0}-\mu)^2}{2\nu}} \mathrm{d}x \\ &= \frac{1}{\sqrt{2\nu\pi}} \int_0^\infty e^{\frac{-(\ln\frac{x}{S_0}-\mu)^2}{2\nu}} \mathrm{d}x.\end{aligned}$$

Changing variable $z = \frac{\ln\frac{x}{S_0}-\mu}{\sqrt{\nu}}$, following by $\mathrm{d}x = \frac{\mathrm{d}x}{x\sqrt{\nu}}$ where $x = S_0 e^{z\sqrt{\nu}+\mu}$

$$\begin{aligned}\mathbb{E}[S_t] &= \frac{S_0}{\sqrt{2\pi}} \int_{-\infty}^\infty e^{\frac{-z^2}{2}} e^{z\sqrt{\nu}+\mu} \mathrm{d}z \\ &= \frac{S_0}{\sqrt{2\pi}} \int_{-\infty}^\infty e^{\frac{-(z-\sqrt{\nu})^2}{2}+\mu+\frac{\nu}{2}} \mathrm{d}z \\ &= \frac{S_0 e^{\mu+\frac{\nu}{2}}}{\sqrt{2\pi}} \int_{-\infty}^\infty e^{\frac{-(z-\sqrt{\nu})^2}{2}} \mathrm{d}z.\end{aligned}$$

Letting $x = z - \sqrt{nu}$

$$\mathbb{E}[S_t] = \frac{S_0 e^{\mu+\frac{\nu}{2}}}{\sqrt{2\pi}} \int_{-\infty}^\infty e^{\frac{-x^2}{2}} \mathrm{d}x = S_0 e^{\mu+\frac{\nu}{2}}. \tag{3}$$

Due to the fact that the last term is equal to $S_0 e^{rt}$ by expected forward price and $\nu = \sigma^2 t$, we get $\mu = \left(r - \frac{\sigma^2}{2}\right) t$. ■

2.2 Brownian motion

First we mentioned a more understandable term, **a random walk**. A symmetric random walk can be defined as at each time point, variable X has an equal chance of increasing or decreasing by 1, that is, $\mathbb{P}(X_{i-1} - X_i = 1) = \mathbb{P}(X_{i-1} - X_i = -1) = \frac{1}{2}$ where $i \in \mathbb{Z}$. Now if we can take smaller and smaller steps in smaller and smaller interval of time, we may get the Brownian motion. That is the scaling limit of random walk in dimension 1.

Brownian motion (or Wiener process) in physics describes the motion of a particle suspended in a fluid. It was first noticed as a model by Robert Brown in 1827 [1], describing "pollen grains suspended in water perform a continual swarming motion". Figure 4 taken from "Brownian Motion - Draft version of May 25, 2008" [2] shows an example of Brownian motion.

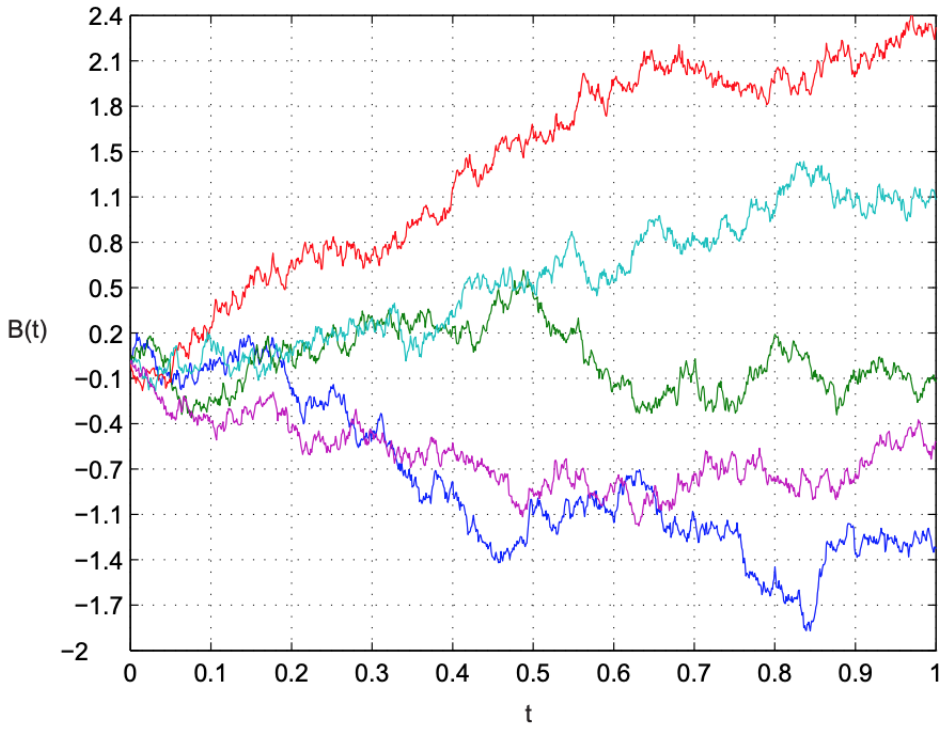


Figure 1: Graph of five sampled Brownian motions from [2]

Definition (Brownian motion).

A one-dimensional (standard) Brownian motion is a real-valued stochastic process B_t, $t \geq 0$ having the following properties:

(a) $B_0 = 0$.

(b) It has independent increments, i.e, if $t_0 < t_1 < ... < t_n$ then $B_{t_1} - B_{t_0}, ..., B_{t_n} - B_{t_{n-1}}$ are independent.

(c) The increment $B_{t+s} - B_t$ has normally distributed with mean 0 and variance s, for all $0 < s, t$.

(d) With probability 1, the function $t \to B_t$ is continuous in t.

Definition (Filtration).

A filtration of the probability space $(\Omega, \mathcal{F}, P)$ is a family $\mathcal{F}(t) : t \geq 0$ of sub-σ-algebras such as $\mathcal{F}(s) \subseteq \mathcal{F}(t)$ for all $s \leq t$ and $\mathcal{F}_\infty = \sigma(\cup_{t \geq 0} \mathcal{F}_t)$.

Recall that a σ-algebra is a family of events including the empty set that is closed under complementation and countable unions. Let $\{B(t)\}$ be a Brownian motion. We denote $\mathcal{F}^0(t)$ the smallest σ-algebra for that each $B(s), 0 \leq s \leq t$ is measurable. Moreover, denote

$$\mathcal{F}^+(t) = \bigcap_{s>t} \mathcal{F}^0(s)$$

and by being right-continuous

$$\bigcap_{\epsilon>0} \mathcal{F}^+(t+\epsilon) = \mathcal{F}^+(t).$$

Definition (Stopping time).
A random variable T in $[0, +\infty]$ is a stopping time with respect to the filtration $\mathcal{F}(t)_{t\geq 0}$ if

$$\{T \leq t\} \in \mathcal{F}(t), \text{ for all } t \geq 0.$$

Definition.
Let T be a stopping time with respect to the filtration $\mathcal{F}^+(t)_{t\geq 0}$. Then

$$\mathcal{F}^+(T) = \left\{A : A \cap \{T \leq t\} \in \mathcal{F}^+(t), \forall t \geq 0\right\}.$$

Theorem 1 (Strong Markov property). *Let $B(t)_{t\geq 0}$ be a Brownian motion and T is a finite stopping time of $B(t)$. Then the process*

$$B(T+t) - B(T) : t \geq 0$$

is also a Brownian motion starting at 0 and independent of $B_0, B_1, ..., B_T$.

Definition (Martingale).
A real-valued stochastic process X_t, $t \geq 0$ is a martingale with respect to a filtration $F(t)$ if $X_t \in F(t)$ for all $t \geq 0$, if $E|X_t| < +\infty$ for all $t \geq 0$, and if $E[X_t|F(s)] = X_s$ almost surely for all $0 \leq s \leq t$.

Lemma 2. *Brownian motion is a martingale.*

Proof. Let $B(t)$ be a standard Brownian motion. Then

$$\begin{aligned} E[B(t)|F^+(s)] &= E[B(t) - B(s)|F^+(s)] + B(s) \\ &= E[B(t) - B(s)] + B(s) \\ &= B(s) \end{aligned}$$

by the Markov property. Hence Brownian motion is a martingale. ■

A random walk, a popular phrase in finance, can be understood as a Brownian motion. It is widely applied to quantitative models due to its properties. The first core reason is that both Brownian motion and assumption of stock prices in finance models have independent movements, that is, the price change does not depend on the past. In other words, the stock price should be unforecastable even all history movements information is provided. In *Proof that properly anticipated prices fluctuate*

randomly [3], Samuelson showed the mathematical foundation to prove that "in well informed and competitive markets, price changes will essentially be random" (Merton, 2006). Second, the Brownian motion's paths are continuous and finite - it is almost certain that it can reach a predetermined target at some time. Moreover, Brownian motion is a Markov process and also a martingale process, which are associated with the features of "efficient markets" mentioned in Farma (1970 [4], 1991 [5]). Markov process is a random process in which if given information until $\tau < t$ then the conditional distribution of $B(t)$ depends only on $B(\tau)$, in the other words, given the present the future is independent from the past. By Lemma 2, Brownian motion is a martingale, so the best estimate of the future value is the current value.

However, there also are critics of the usefulness of Brownian motion in mimicking the stock price in financial models. Based on the core assumptions of the models which use Brownian motion, namely independence, stationarity and normal distribution, Borna and Sharma (2011) pointed out features that Brownian-motion-based-models were not suitable in the real-world markets [6].

2.3 The Itô process

Itô calculus is named after Kiyoshi Itô, it uses the methods of calculus applied to stochastic processes. Therefore, it plays an important role in mathematical finance and stochastic differential equations. We, however, will only stress some main points of the Itô process and Itô's Lemma.

The Itô process is defined as the integral equation

$$S(t,w) = S(0) + \int_0^t \mu[u, S(u,w)]du + \int_0^t \sigma[u, S(u,w)]dW(u,w) \tag{4}$$

or as a stochastic differential equation

$$dS(t,w) = \mu[t, S(t,w)]dt + \sigma[t, S(t,w)]dW(t,w). \tag{5}$$

The Itô equation is a continuous-time random equation and its domain is $[0,\infty)\times\Omega$, where the two arguments t and w represent time and random element respectively. The equation (5) in finance can be explained as the expression of small change in the stock price $S(t,w)$ at time t affected by the random element w. The small difference $dS(t,w)$ is the limit of $\Delta S(t,w)$ as Δt approaches 0, where $\Delta S(t,w) = S(t+\Delta t,w) - S(t,w)$ and Δt is the difference in time. The drift component $\mu[t, S(t,w)]$ computes the expected change in $S(t,w)$. The term $\sigma[t, S(t,w)]dW(t,w)$ represents the uncertainty of $dS(t,w)$ where $\sigma[t, S(t,w)]$ is used to calculate the standard deviation or the volatility of $dS(t,w)$

and $dW(t, w)$ is Brownian motion with mean 0 and variance dt. A more detailed explanation about the above uncertain term can be found in Chapter *Ito's Calculus and the Derivative of the Black-Scholes Option-Pricing Model* from the book *Handbook of Quantitative Finance and Risk Management* [7]. Also, in this book, the authors give a detailed introduction and calculation of Itô calculus and how its importance to models of asset pricing.

Itô's lemma is the chain rule for stochastic calculus. We can state the Itô's lemma as follows:

Theorem 2 (Itô's lemma). *Let $W(t)$ be a Wiener process and $S(t, w)$ be an Itô drift-diffusion process which satisfies the stochastic differential equation:*

$$dS(t, w) = \mu(t, S(t, w))dt + \sigma(t, S(t, w))dW(t, w). \tag{6}$$

If $f(t, w) \in C^2(\mathbb{R}, \mathbb{R}^2)$ then $f(t, S(t, w))$ is also an Ito drift-diffusion process, with its differential given by:

$$df = \left[\frac{\partial f}{\partial t} + \mu\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2\frac{\partial^2 f}{\partial S^2}\right]dt + \sigma\frac{\partial f}{\partial S}dW. \tag{7}$$

The result (7) can be derived easily by using Taylor expansion of $df(t, S(t, w))$

$$df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial S}dS + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}dS^2 + ... \tag{8}$$

Substituting the stochastic differential equation (6) into Taylor expansion (8)

$$df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial S}(\mu dt + \sigma dW) + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}(\mu^2 dt^2 + 2\mu\sigma dt dW + \sigma^2 dW^2) + ... \tag{9}$$

Finally, setting dt^2 and $dtdW$ be equal to 0, dW^2 be equal to dt (by the quadratic variance of a Wiener process) we get (7).

More about derivation of this Itô's lemma and its extension are presented in [17] - [20].

2.4 Greek letters

Under the Black-Scholes model framework [23], option traders usually use Greek letters to measure different dimensions to the risk of an option. Often-mentioned Greek letters of Delta, Theta, Gamma, Vega, and Rho in option pricing are generally defined as the sensitives of an option price relative to changes in the value of either a state variable or a parameter (Hull, "Options, Futures, and Other Derivatives" [15]). Based

on our need in this paper, we will only focus on some of the letters. This section has used materials from [15], [21] and [22].

Definition.

The **Delta** of an option measure the rate of change of option value to the change in the underlying asset's price.

$$\text{Delta} = \Delta = \frac{\partial V}{\partial S_0}.$$

The **Vega** of an option is the sensitivity of the option price to a change in volatility of the underlying stock.

$$\text{Vega} = \nu = \frac{\partial}{\partial \sigma} V,$$

where the parameter V denotes the option's value, either C for call option or P for put option, and S_0 denotes the current price of the underlying asset.

In the book *On Derivatives of Black-Scholes Greek Letters* (2013) [21], authors gave the detailed proof of the following two lemmas.

Lemma 3. *From the relationship of* d_1 *and* d_2

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)\tau}{\sigma\sqrt{\tau}}, \text{ and } d_2 = d_1 - \sigma\sqrt{\tau},$$

it holds that

$$\frac{\partial d_2}{\partial \sigma} = \frac{\partial d_1}{\partial \sigma} - \sqrt{\tau}. \tag{10}$$

Here the parameter τ is the time from current time to the maturity.

Lemma 4. *The relationship between the values of the density functions* $n(d_1)$ *and* $n(d_2)$ *can be expressed as*

$$S_0 n(d_1) = K e^{-r\tau} n(d_2). \tag{11}$$

With the two above relationships, the formula of the Vega letter can be derived.

Proposition 1. *The expression of Vega letter (Vega function) and Delta letter for Black-Scholes call options is*

$$\Delta = \frac{\partial C}{\partial S_0} = N(d_1), \tag{12}$$

$$\nu = \frac{\partial C}{\partial \sigma} = \sqrt{\tau} S_0 n(d_1). \tag{13}$$

Proof. To prove this proposition, we will need the formula of the call option price $C = S_0 N(d_1) - K e^{-r\tau} N(d_2)$, which is derived from [23] and will be mentioned later

in this paper. Taking the derivative with respect to S_0 directly gives us the equation (12).

On the other hand, taking derivative with respect to σ

$$\begin{aligned}
\nu = \frac{\partial C}{\partial \sigma} &= \frac{\partial [S_0 N(d_1) - K e^{-r\tau} N(d_2)]}{\partial \sigma} \\
&= S_0 n(d_1) \frac{\partial d_1}{\partial \sigma} - K e^{-r\tau} n(d_1) \frac{\partial d_2}{\partial \sigma} \\
&= S_0 n(d_1) \frac{\partial d_1}{\partial \sigma} - K e^{-r\tau} n(d_2) \left[\frac{\partial d_1}{\partial \sigma} - \sqrt{\tau} \right] \\
&= \left[S_0 n(d_1) - K e^{-r\tau} n(d_2) \right] \frac{\partial d_1}{\partial \sigma} + \sqrt{\tau} K e^{-r\tau} n(d_2) \\
&= \sqrt{\tau} K e^{-r\tau} n(d_2),
\end{aligned}$$

where the third last equation comes from (10) and from (11) we derive $S_0 n(d_1) - K e^{-r\tau} n(d_2) = 0$ which leads to the last equation. ■

The normal density function is given by

$$n(x) = \tfrac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2},$$

where μ is the mean and σ is the standard deviation of the distribution.

Assume that d_1 is standard normal distributed, then

$$n(d_1) = \tfrac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}d_1^2}$$

and Equation (13) becomes

$$\nu = \frac{1}{\sqrt{2\pi}} S_0 \sqrt{\tau} e^{-\frac{1}{2}d_1^2}. \tag{14}$$

3 Models of asset pricing

This section presents two famous developments of option pricing construction and model. The interesting point is that both models discuss market price behavior in continuous time and apply stochastic analysis of Brownian motion. The first model of Bachelier is derived from arithmetic Brownian motion while the second model, the Black-Scholes model, was built upon the consideration of geometric progression arriving at geometric Brownian motion.

3.1 The Bachelier Model (1900)

Louis Jean-Baptisete Alphonse Bachelier (1870 - 1946) was the first mathematician to use Brownian motion as a model to analyze stock and option market prices in 1900 [25].

He modeled the stock price as normally distributed:

$$S_t = S_0(1 + \sigma W_t),\ W_t \sim N(0, t).$$

This is considered to be a good model for interest rates. However, it leads to non-zero probability for negative stock price.

The Bachelier model for the stock price follows the stochastic differential equation:

$$dS_t = \mu S_t dt + \sigma dW_t,$$

where μ and $\sigma > 0$ are constants and W_t is Brownian motion.

3.2 The Black-Scholes Model (1976)

Black-Scholes model (fully Black-Scholes-Merton model) was proposed first in 1973 by Fischer Black, Myron Scholes and Robert Merton [23] - [24].

This model can be applied to determine the price of European options with risk-less interest rate r, where the underlying is non-dividends. We will discuss in more details about the assumptions of Black-Scholes model and how much it can be trusted in actual market later in this section.

In this section, we still use $S(t, w)$ as the notation for the stock price at time t, uncertain term w and the stock price is assumed to follow the Itô's stochastic differential equation (5). Then the stock option at time t can be written as

$$V(t, w) = V(t, S(t, w)) \tag{15}$$

and satisfies that $V(t, w)$ is twice continuously differentiable.

For simplifying purposes, we only consider the case $\mu[t, S(t, w)] = \mu S(t, w)$ and $\sigma[t, S(t, w)] = \sigma S(t, w)$, where μ and $\sigma > 0$ are constants. We also simplify the notation by getting rid of the uncertain term w in the expressions. Thus, the equation (5) changes into

$$dS_t = \mu S_t dt + \sigma S_t dW_t. \tag{16}$$

Recall S is the price of non-dividend paying asset, W is a Wiener process, the time t, the drift parameter μ and the volatility parameter σ.

Applying **Theorem 2 (Itô's lemma)** to equations (15) and (16), we have

$$dV = \left[\frac{\partial V}{\partial t} + \mu S_t \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2}\right] dt + \sigma S_t \frac{\partial V}{\partial S} dW_t. \tag{17}$$

Now we will apply a technique called Delta-Hedging. In this technique, we denote the quantity of the asset Δ. Hence, the change of a mixture of option value and quantity of assets in time is

$$d(V + \Delta S_t) = \left[\frac{\partial V}{\partial t} + \mu S_t \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2} + \Delta \mu S_t\right] dt + \sigma S_t \left(\frac{\partial V}{\partial S} + \Delta\right) dW_t \tag{18}$$

Choose $\Delta = -\frac{\partial V}{\partial S}(t, S_t)$, equation (18) becomes

$$d(V + \Delta S_t) = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\right) dt. \tag{19}$$

This technique provides us with a portfolio that does not contain the random term W_t. Hence, the growth rate of this delta-hedging portfolio (19) is equal to the compound risk free rate r. Thus

$$\left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\right) dt = r(V + \Delta S_t) = r\left(V - S_t \frac{\partial V}{\partial S}\right). \tag{20}$$

Rearrange the equation (20) we get the famous Black-Scholes PDE

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2} + r\frac{\partial V}{\partial S} - rV = 0. \tag{21}$$

A very nice and proper approach to Black-Scholes PDE from portfolio value equation can be seen at [7].

Theorem 3 (Black Scholes formula). *Consider in a risk-neutral universe with S_0 and S_t are stock price at initial and at time t respectively, S_t is log-normal distributed. At time $t = 0$, with maturity time T, strike price K and risk-free rate r, a Vanilla European call option price is given by*

$$C = S_0 N\left(\frac{\log \frac{S_0}{K} + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\right) - Ke^{-rT} N\left(\frac{\log \frac{S_0}{K} + (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\right), \tag{22}$$

with the cumulative normal distribution

$$N(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} exp\frac{-z^2}{2} dz.$$

Other notations are often used $d_1 = \dfrac{\log \frac{S}{K} + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}$ and $d_2 = d_1 - \sigma\sqrt{T}$. Then in short,

$$C = S_0 N(d_1) - K e^{-rT} N(d_2). \tag{23}$$

To prove the call option pricing formula we will calculated the expected option value. In fact, it is not the original method that derived from [23].

Recall that the stock price S_t is *log-normally distributed* means that $\log S_t$ is normally distributed.

Proof. We know that

$$\begin{aligned} C(T,S) &= S_T - K && \text{if } S_T > K \\ &= 0 && \text{if } S_T \leq K \end{aligned}$$

By the definition of a risk-neutral universe,

$$\begin{aligned} C(0,S) &= e^{-rT}\mathbb{E}[C(T,S)] \\ &= e^{-rT}\mathbb{E}[\max(S_T - K, 0)] \\ &= e^{-rT}\int_K^\infty \frac{1}{\sqrt{2\pi T}\sigma x}(x-K) e^{\frac{-(\ln \frac{x}{S_0} - \mu)^2}{2\sigma^2 T}} \mathrm{d}x \\ &= e^{-rT}\frac{1}{\sqrt{2\pi T}\sigma}\int_K^\infty e^{\frac{-(\ln \frac{x}{S_0} - \mu)^2}{2\sigma^2 T}} \mathrm{d}x - e^{-rT}\int_K^\infty \frac{1}{\sqrt{2\pi T}\sigma x} K e^{\frac{-(\ln \frac{x}{S_0} - \mu)^2}{2\sigma^2 T}} \mathrm{d}x. \end{aligned}$$

The first integral we encountered in Lemma 1, thus it can be written as

$$e^{-rT} S_0 e^{\mu + \frac{\sigma^2 T}{2}} \int_{\frac{\ln \frac{K}{S_0} - \mu - \sigma^2 T}{\sigma\sqrt{T}}}^\infty \frac{1}{\sqrt{2\pi}} e^{\frac{-z^2}{2}} \mathrm{d}z.$$

Substitute equation (2) for value μ and see that it is indeed the cumulative normal distribution for variable $\frac{\ln \frac{K}{S_0} - rT - \frac{\sigma^2 T}{2}}{\sigma\sqrt{T}}$. Then

$$\begin{aligned} S_0\left(1 - N\left(\frac{\ln \frac{K}{S_0} - rT - \frac{\sigma^2 T}{2}}{\sigma\sqrt{T}}\right)\right) &= S_0 N\left(-\frac{\ln \frac{K}{S_0} - rT - \frac{\sigma^2 T}{2}}{\sigma\sqrt{T}}\right) \\ &= S_0 N\left(\frac{\ln \frac{S_0}{K} + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\right), \end{aligned}$$

which is also the first term of (22).

For the second term, we use change of variable $y = \frac{\ln \frac{x}{S_0} - \mu}{\sigma\sqrt{T}}$, $\mathrm{d}y = \frac{\mathrm{d}x}{x\sigma\sqrt{T}}$ and let $A = \frac{\ln \frac{K}{S_0} - \mu - \sigma^2 T}{\sigma\sqrt{T}}$. Then

$$\begin{aligned}
-e^{-rT}\int_K^\infty \frac{1}{\sqrt{2\pi T}\sigma x} K e^{\frac{-(\ln \frac{x}{S_0} - \mu)^2}{2\sigma^2 T}} \mathrm{d}x &= -e^{-rT}\int_{A+\sigma\sqrt{T}}^\infty \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} \mathrm{d}y \\
&= -e^{-rT} K \left(1 - N\left(A + \sigma\sqrt{T}\right)\right) \\
&= -Ke^{-rT} N\left(-A - \sigma\sqrt{T}\right) \\
&= -Ke^{-rT} N\left(\frac{\ln \frac{S_0}{K} + (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\right),
\end{aligned}$$

which is what we need. ■

Despite this model is used widely, Black-Scholes model does not exactly follow the actual market. Following are assumptions of this model

- The model only examine the European-style options which can only exercised on the expiration date.
- No dividends: The model assumes that the stocks do not pay any dividends or returns.
- Frictionless market: There is no transaction cost or services cost of buying or selling options.
- Normal distribution: The returns on the underlying stock is normally distributed. It implies that the volatility of the market is a constant.
- Risk-free interest rate: The short-term interest rate is known and constant.
- No arbitrage: There is no arbitrage. It avoids the opportunity of making a riskless profit.

Due to some of those impractical assumptions, the Black-Scholes model has to face with certain limitations in its ability to predict option prices. In [30], Krznaric presents a project that analyzes the price movement of 480 stocks during the year 2014 to determine the effectiveness of Black-Scholes model. Also in [31] authors criticize the unrealistic of applying Black-Scholes formula.

4 Volatility

Portfolios are designed with the understanding that risk and return are closely correlated. This implies that the higher the yield an investor seeks, the higher the risk they must take. One of the main principles of modern portfolio theory is that financial results can be taken in account in terms of mean (the return) and standard deviation of the return (the risk) of a given security. One of the biggest difficulties that investors face is determining how much risk they are taking on as they accept an investment. That means it is important that people somehow can define the future trends of portfolio components. To have information about the future, in many aspects of life, one efficient way is to investigate the historical statistic performance. However, in social sciences such as finance, making this inference regarding past results might not be the safest way for an investor to estimate risk. There is a possibility that past forecasts will result in either too much or far too little risk for a specific investor. **Volatility** is one estimator often used to evaluate how well one portfolio performs and in this section, we will approach this term in two aspects: historical volatility and implied volatility.

4.1 Historical Volatility

Historical volatility is based on historical data like prior price movement. Therefore, it is based on actual documentation of the stock's past performance. To compute historical volatility, we need the asset price data at times t_i with $t_i - t_{i-1} = \Delta$ and $Y_i = \ln S_{t_i} - \ln S_{t_{i-1}}$ (log-return). Estimated historical volatility is $\hat{\sigma}_{hist} = \frac{\hat{\sigma}_\Delta}{\sqrt{\Delta}}$ where

$$\hat{\sigma}_\Delta = \left(\frac{1}{N-1} \sum_{i=1}^{N} (Y_i - \overline{Y})^2 \right)^{\frac{1}{2}} \text{ and } \overline{Y} = \frac{1}{N} \sum_{i=1}^{N} Y_i.$$

Historical volatility is a statistical term to measure how far the stock price moves away from the average price in a given period of time. In general, historical volatility is simply based on the collected observations, it provides the trader the information about the past trends rather than a future expectation movements. A high historical volatility may indicate that the price of the financial commodity has been rapidly rising and falling over time but has not changed dramatically from its initial price. Similarly, low historical volatility may mean that the price of a financial product has not moved much but has moved slowly in a direction. By this definition, we can see similarity between historical volatility of stock price and standard deviation. Therefore, using the formula of standard deviation is the most common way to compute historical volatility, however, not the only way.

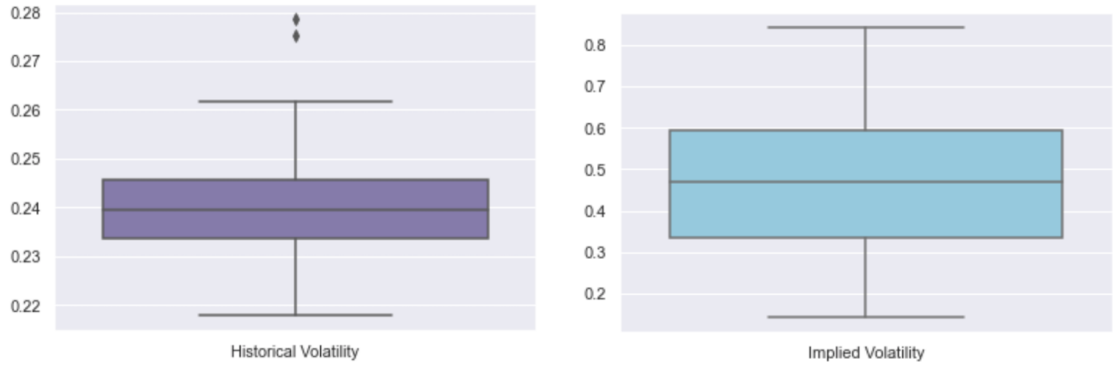
	contractSymbol	lastTradeDate	strike	ask	volume	impliedVolatility	inTheMoney	currency	maturityTime	Date	Close
0	AAPL220121C00027500	2021-05-13	27.50	98.80	2.0	0.822267	True	USD	253	2021-05-13	124.970001
1	AAPL220121C00028750	2021-05-17	28.75	97.60	2.0	0.828127	True	USD	249	2021-05-17	126.269997
2	AAPL220121C00030000	2021-05-07	30.00	96.35	3.0	0.804689	True	USD	259	2021-05-07	130.210007
3	AAPL220121C00031250	2021-05-06	31.25	95.10	991.0	0.783205	True	USD	260	2021-05-06	129.520004
4	AAPL220121C00032500	2021-05-07	32.50	93.85	9.0	0.761721	True	USD	259	2021-05-07	130.210007
...	...	...	...	...	...	...	...	...	...	...	...
194	AAPL211119C00160000	2021-05-18	160.00	1.82	224.0	0.291023	False	USD	185	2021-05-18	124.849998
195	AAPL211119C00165000	2021-05-18	165.00	1.43	40.0	0.294990	False	USD	185	2021-05-18	124.849998
196	AAPL211119C00170000	2021-05-18	170.00	1.13	245.0	0.299079	False	USD	185	2021-05-18	124.849998
197	AAPL211119C00175000	2021-05-17	175.00	0.93	22.0	0.305671	False	USD	186	2021-05-17	126.269997
198	AAPL211119C00180000	2021-05-18	180.00	0.78	2.0	0.312751	False	USD	185	2021-05-18	124.849998

199 rows × 11 columns

Table 1: Sample of Apple option trading data fetching from Yahoo Finance

Table 1 shows a sample of Apple options market that we can fetch from Yahoo Finance. Call option prices are collected, here we use the *ask* price, together with data of strike price *strike*, time to maturity *maturityTime*, date of option trading *Date* and the closing stock price on that date which in the column *Close*.

To compute the daily historical volatility we use the log-return price sample of size 21 days (average number of trading days in one month) and compute for each trading day. To compute annual historical volatility we simply take daily volatility multiply with square root of 252. This annual historical volatility will be compared with the volatility that traders used in reality for the in the money call options that expiring within one year, which is provided by Yahoo Finance in column *impliedVolatility*. However, the volatility differed in various contracts even those contracts had the same trading time. We, among different data in one same day, only consider the lowest volatility which is greater than 10^{-5}.



(a) Historical volatility

(b) Implied volatility

Figure 2: Historical volatility in comparison with volatility used for trading

The statistical description of historical volatility and realistic trading volatility is

given in Figure 2. As we can observe, the historical estimate significantly underestimates the volatility used by most traders. The maximum value of annual historical volatility in this sample is slightly below the first quartile of realistic volatility. The difference of mean absolute deviation (MAD) of historical and realistic volatility is 0.11. Since historical volatility is the standard deviation of log returns from the previous year, it is not shocking that the calculation smoothes out the peaks, resulting in a value for each day that is less volatile and thus less sensitive to daily market movements. Thus, traders need a different method to calculate volatility which generates values only from trading information on that day but not a previous time period. That alternative variable is implied volatility.

4.2 Implied Volatility

Black-Scholes Volatility

Unlike historical volatility, implied volatility is the estimated volatility of a stocks calculated by the price of an option on that stock. Hence, the volatility is *implied* by the financial model. Recall the Black-Scholes model, the volatility coefficient is assumed to be constant with respect to the strike price and the maturity. In reality, it is also a limitation of Black-Scholes volatility. In [15], there is discussion that implied volatility varies with strike price (creating a *volatility smile*) and maturity (creating a *volatility term structure*). One other way to avoid that limitation is to model the volatility as a diffusion process - the stochastic volatility models [26] or the autoregressive conditional heteroscedasticity (ARCH) model.

While most of the literature consider implied volatility being better than historical volatility, author Isaac Faber from Stanford University used hypothesis testing on S & P 500 index to conclude the failure to reject the hypothesis that the two kinds of volatility are the same [59]. Basically, when it put into practice, we may assume that implicit volatility acts better at times and worse at others, but which times are which is unclear.

Let us consider a call option of value $C(t, S)$ with strike price K and maturity T is traded at time t and at a give stock price (S_t). Assume that the option value is calculated by the Black-Scholes model. Then the implied volatility $\hat{\sigma}_{imp}$ is given by the solution of equation

$$C_{BS}(S_t, \hat{\sigma}_{imp}, K, T) = C(\hat{\sigma}_{imp}). \tag{24}$$

It is well known that there is no expression for the Black - Scholes implied volatility. In the paper *Can there be an explicit formula for implied volatility?*, authors considered

the implied volatility as a function of underlying, strike and call price and proved that this function did not belong to the class of D-finite functions.

Using the same notations as above, assuming a fixed maturity $T > 0$ throughout, we omit T then the implied volatility is considered as the function I which satisfies

$$C_{BS}(S_t, I(S_t, K, C), K) = C(t, S) \tag{25}$$

and is defined on the open set

$$D_I := \left\{(S_t, K, C) \in \mathbb{R}^3 : S, K > 0, (S-K)^+ < C < S\right\}.$$

Since the Black - Scholes call price is real analytic for $S, K, \sigma > 0$ and the Black - Scholes Vega is positive, the implicit function theorem shows that I is real analytic on D_I. The question is concerned is that whether the function I admits a closed form. To give a partial answer, author contributed Theorem 4 as follow

Theorem 4. *The function I: $D_I \subset \mathbb{R}^3 \to (0, \infty)$ defined by (25) is not D-finite.*

To prove the theorem, author used the closure under algebraic substitution of D-finite functions. Suppose that a C^∞-smooth function f is defined on an open set $D_f \subset \mathbb{R}^n$. It is called D-finite if it satisfies PDEs

$$\begin{aligned}
P_{1,d_1}(\mathbf{x})\frac{\partial^{d_1} f(\mathbf{x})}{\partial x_1^{d_1}} + P_{1,d_1-1}(\mathbf{x})\frac{\partial^{d_1-1} f(\mathbf{x})}{\partial x_1^{d_1-1}} + ... + P_{1,1}(\mathbf{x})\frac{\partial f(\mathbf{x})}{\partial x_1} + P_{1,0}(\mathbf{x}) f(\mathbf{x}) = 0, \\
\vdots \\
P_{n,d_n}(\mathbf{x})\frac{\partial^{d_n} f(\mathbf{x})}{\partial x_n^{d_n}} + P_{n,d_n-1}(\mathbf{x})\frac{\partial^{d_n-1} f(\mathbf{x})}{\partial x_n^{d_n-1}} + ... + P_{n,1}(\mathbf{x})\frac{\partial f(\mathbf{x})}{\partial x_n} + P_{n,0}(\mathbf{x}) f(\mathbf{x}) = 0,
\end{aligned}$$

valid for $\mathbf{x} = (x_1, x_2, ..., x_n) \in D_f$, where $d_i \geq 1$ for $i = 1, ..., n$ and the P_{ij} are polynomials such that P_{i,d_i} is not identically zero on D_f for $i = 1, ..., n$. If f is real analytic, fix an arbitrary point $\mathbf{x_0} \in D_f$ and consider the Taylor expansion of f at $\mathbf{x_0}$. If we view f as formal power series, then the above PDEs shows that its partial derivatives generate a finite dimensional vector space over the field of rational functions. [28], [29]

The class of D-finite functions is closed under addition, multiplication, (in-)definite intergration and Laplace transform. Division does not preserve D-finiteness in general, nor does composition unless the inner function is algebraic.

The theorem does not rule out all explicit expressions but it shows that implied volatility does not belong to a certain large class, which contains many elementary functions and classical special functions.

Therefore, in order to get $\hat{\sigma}_{imp}$ we normally need to solve the Black-Scholes equations for implicit roots.

Newton Raphson Method for computing Implied Volatility

One way to solve Black-Scholes formula is to use Newton Raphson method (also known as Newton method) - an iterative algorithm to estimate the nonlinear equation's roots $f(x) = 0$, where $f(x)$ is assumed continuous and differentiable. If we know the root we are looking for is near $x = x_0$ then the Newton method tells us that we can do a better approximation by

$$x_1 = x_0 - \frac{f(x_i)}{f'(x_i)}.$$

This procedure can be repeated as many times as needed to achieve the desired precision. Formula for the $(i+1) - th$ iteration is given by

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

Newton method is implemented based on the idea that a continuous and differentiable function can be approximated by a straight line tangent to it.

Now assume the function C_{BS} in Black-Scholes equation (24) is continuous and differentiable with respect to $\hat{\sigma}_{imp}$, we can use Newton method to find the root of equation. Applying to formula (24) to find $\hat{\sigma}_{imp}$, we get:

$$\hat{\sigma}_{i+1} = \hat{\sigma}_i - \frac{C(\hat{\sigma}_i)}{\frac{\partial}{\partial \hat{\sigma}_i} C},$$

where $C(\hat{\sigma}_i)$ is the market value of an option, $\frac{\partial}{\partial \hat{\sigma}_i} C$ is the Vega function.

Substituting the expression of Vega function (14) into the above iteration we can get fully the equation used to update volatility after each iteration.

$$\hat{\sigma}_{i+1} = \hat{\sigma}_i - \frac{(C(\hat{\sigma}_i) - C)\sqrt{2\pi} e^{1/2 d_1^2}}{S_0 \sqrt{T}}$$

where all notations are the same as in Black-Scholes formula and C is the trading call option.

The Newton - Raphson method needs an initial guess which is σ_0 for the first update. In certain cases, the method's ability to converge may be influenced by the choice of initial guess. Newton method may not work if there are points of inflection, local maxima or minima around σ_0 or the root.

We will simply try Newton method which uses exactly the computation above with 2 examples. The input includes 5 values which are stock price S, strike price K, maturity time T, risk-free rate r and a predetermined volatility σ, the call option price is computed by Black-Scholes formula then all variables play role as input of the

iteration method with initial guess of σ is 0.9. Newton method will stop computing at i-th iteration and return σ_i as an approximation of implied volatility if the difference between σ_i and σ is below 1.0e-05 or return *Error* if the value of vega function at some iteration reaches 0. The source code to apply the iteration method is generated by the author and implemented by Python (version 3.8.3).

$$\text{Input 1 } (S, K, T, r, \sigma) = (100, 100, 10, 0.01, 0.45)$$
$$\text{Input 2 } (S, K, T, r, \sigma) = (170, 30, 4, 0.04, 0.71)$$

Input	BS Call option price	Newton Method		
		Iteration	Vega value	Volatility
Input 1	147.76	1	43.57	0.19788
		2	112.82	0.43059
		3	94.92	0.44979
		4	92.95	0.44999
Input 2	54.69	1	23.93	6.67536
		2	1.1e-08	$>$ 1.0e+10
		3	0.0	Error

Table 2: Results of applying Newton method with 2 input examples

Newton method can perform a good convergence process, much faster than other methods such as bisection method, since it has quadratic convergence. However, it requires calculating the derivative, in our case is the Vega function. In some of the cases, this value can be near zero then causes the dividing by zero problem as shown in the above example. This also shows the instability convergence property of Newton method. In some cases, the dividing by zero problem can be solved only by choosing another initial point. But it is not always the case, it depends much on the data.

A call option is called **In The Money (ITM)** if the current market price is higher than the strike price predetermined in the option. In the other hand, it is **Out of The Money (OTM)** when the current market price is lower than the strike price, which means the holders can not make any profit by exercising the option. They are two classes in **moneyness**, moneyness value can be determined by the ratio between the strike price and stock price. Figure 3 below shows the relationship of moneyness and vega. When the data is too deep ITM or OTM, the Vega value can be close to zero. In 2006, Peter Jackel wrote a paper about problems in methods of solving implied volatility [68].

Moreover, running time is also an aspect which needs to be concerned, whenever a sample is improved by updating more new data, Newton method needs to be run the

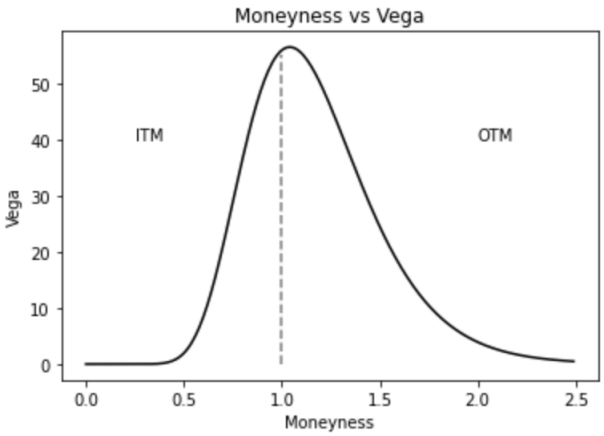


Figure 3: Moneyness versus Vega

whole process again. Even without the cases of Error, this process is still very time consuming since the large number of data needed.

One method we want to introduce in this thesis is applying neural networks to solve this situation. At first it seems like to solve the problem when updating data since we can use the already trained model to apply to predict for other sample so using neural network is more applicable in this sense.

5 Neural Network

People now more and more feel familiar with the terminology *machine learning* or *deep learning*. In general, machine learning uses algorithms on data, to learn and make informed decisions based on it. Deep learning is more complex. It structures algorithms into layers called *artificial neural network* that can learn and make intelligent decisions on its own. In this section, we will focus on *neural networks*, the *neuron* we consider is not biological neuron. In fact, they are artificial models being inspired by the model of biological neurons and built with ambition of mimicking the way human brain's solving cognitive tasks.

5.1 History

The concept and idea about an artificial neural networks had raised from the late 19th centuries by some scientists. Back to this time, scientists emphasized on constructing general theories of learning based on physics, psychology and neurophysiology works, however, did not have any mathematical models.

The modern view of neural networks began in 1940s with the work [34] of Warren McCulloch and Walter Pitts and was followed by Donald Hebb [35]. The invention of perceptron network associated with learning rule [36] is known as the first practical application of artificial neural networks. In this paper, Bernard Widrow and Ted Hoff introduced a new learning algorithm and used it to train adaptive linear neural networks. This Widrow-Hoff learning rule is still applicable in some today's models. Many important works came after the two pioneers focusing on new aspects that neural networks can develop such as memories [37] - [38] or self-organizing property [39]. After a stagnating period because of the lack of new ideas and the limitation of computers, neural networks increased dramatically again in 1980s with two new concepts.

The first concept was to use statistical mechanics to explain the operation of a certain class of recurrent network [40]. The second was the backpropagation algorithm for training multilayer perceptron networks by David R. and James M. [41]. Many progresses of neural networks had been made based on this two new concepts and followed by a large number of applications.

Neural networks clearly do not provide solutions to every problem, but they are essential tools to be used in appropriate situations (*Neural Network Design*, 2014 [33]). People can not deny the important position of neural networks in modern science. However, since our knowledge of human brain is very little, the most advances in neural networks still have not yet to come.

5.2 Neuron model and Network architectures

Neuron Model

A neuron model can be single-input or more typically multiple-input. In case of multiple-input neuron, assume there are d inputs $p_1, p_2, ..., p_d$ in the form of a vector p. Each input $p_1, p_2, ..., p_d$ is weighted by corresponding parameters $w_{11}, w_{12}, .., w_{1d}$ which form the weight matrix W_1. The input vector $p \in \mathbb{R}^{d\times 1}$ is multiplied by the weight matrix $W_1 \in \mathbb{R}^{1\times d}$ then added with the bias $b \in \mathbb{R}$, to form the net input:

$$n = W_1 p + b. \tag{26}$$

The net input n goes through an activation function f to produce the neuron output x

$$x = f(W_1 p + b) \tag{27}$$

The sense of weight matrix and activation function will be discussed later after we review the Network architecture.

Network Architectures

A number d_1 of neurons operating in parallel is called a **layer** (figure below). Then with an input $p \in \mathbb{R}^{d\times 1}$ a layer includes the weight matrix $W \in \mathbb{R}^{d_1\times d}$, a bias $b \in \mathbb{R}^{d_1}$, the activation function f and the output after that layer $x \in \mathbb{R}^{d_1\times 1}$. Also the activation function is applied for each neuron in a layer can be different as well.

From the architecture of one layer, we can easily construct a network with multi-layer (figure below). The $j-th$ layer will have its own weight matrix $W_j \in \mathbb{R}^{d_j\times d_{j-1}}$, bias $b \in \mathbb{R}^{d_j}$, the activation function f_j and the output after that layer $x_j \in \mathbb{R}^{d_j\times 1}$. Only an input can be called **input layer**. A layer whose output is the network output is called **output layer**. The other layers are called **hidden layers**. For example, a three-layer network will contain an output layer and other two hidden layers, its ultimate output can be computed by

$$x_3 = f_3(W_3 f_2(W_2 f_1(W_1 p + b_1) + b_2) + b_3.$$

In general,

$$x^{(j)} = f\left(W_j x^{(j-1)} + b_j\right), \tag{28}$$

where $x^0 = x$ the input, the matrix $W_j \in \mathbb{R}^{d_j\times d_{j-1}}$ and vector $b_j \in \mathbb{R}^{d_j}$ are parameters of $j-th$ layer, and f is the activation function. Equation (28) clearly shows that the

output of the previous layer plays a role as an input for the next layer. Also, the ultimate output vector after J layers is x^J and also represented for the prediction function $h(x; w)$ where w contains all parameters $(W1, b1), ..., (W_J, b_J)$ [32]. When building a model, our target usually is to minimizing the *distance* from our model predicted value $h(x; w)$ and the true value y. This *distance* is measured by the *loss function* l, then the optimization problem with given training set $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ leads to

$$\min \frac{1}{n} \sum_{i=1}^{n} l(h(x_i, w), y_i). \tag{29}$$

Normally, the choice of loss function directly depends on the activation function used in the model.

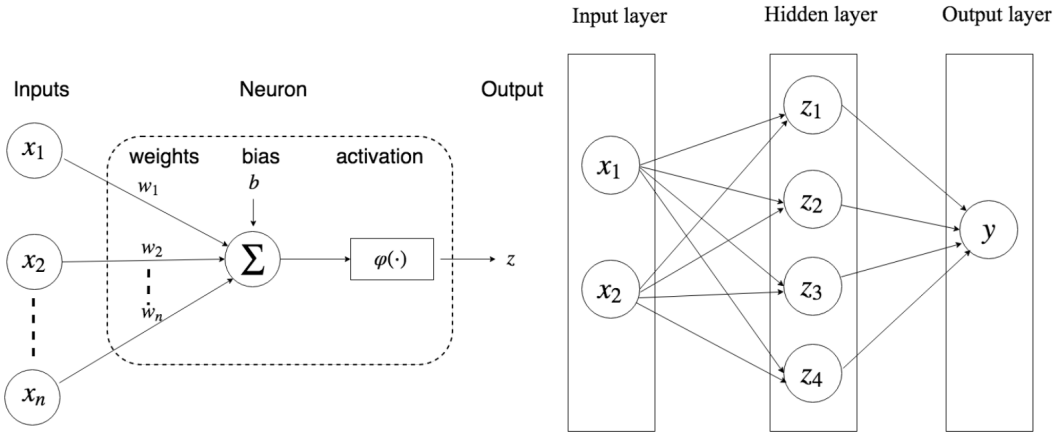


Figure 4: Example of Neural network architecture [53]

If only consider the *net input*, it can be constructed linearly from given inputs. However, it can reach arbitrarily high or low values and lead to computational issues in deep neural network. Then the activation function f is added to give a certain limitation for a layer's output before it goes through the next layer. In addition and also more importance, an activation function gives the ability to add the non-linearity into network. The certain perceptual tasks, which need the participation of deep neural networks to solve, in general do not have linear patterns. There are many activation functions, each solves one feature based on needs of the models. Below are some examples of activation functions

- The sigmoid function $f(x) = 1/(1 + exp(-ax))$
- The ReLU (rectified linear unit) function $f(x) = \max\{0, x\}$
- The Leaky ReLU function $f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}$

5.3 Hyper-parameters

To build a good deep neural network model, we need to involve various choices of *hyper-parameters*.

Batch size is the number of samples that will go through the network at once, before update the internal parameters of the model. A training dataset can be separated into one or more batches. For example, the dataset contains 2000 samples and the batch size is set up equal to 150. That means, model will take the first 100 samples from training dataset and train. Next it will take the second 150 samples and train again the network which has updated all parameters after the first batch, keep doing this process until all samples go through the network. By this way, our process requires less memory since each time training network, we only need to use one portion, not all, of the training dataset. Moreover, the typically model trains faster and updates parameters more efficiently. On the other hand, dividing training dataset into many batches may cause less accuracy in computing the gradient.

The number of *epochs* defines the number of times that algorithm works with entire training dataset. The number of epochs in general is large, allowing the algorithm to run until reach some sufficient small loss. As in Figure 5, loss function used is the mean absolute error (mae), value of the loss has a huge decline only after epoch 40 and gradually decreasing trend after that.

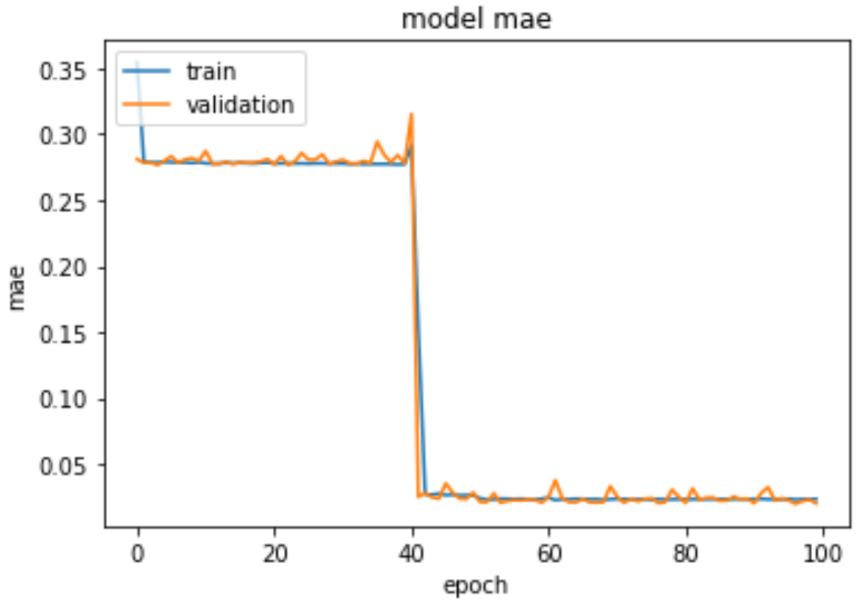


Figure 5: Change of loss value with respect to number of epochs [1]

[1] All figures in this section are generated by different models using the datasets which will be described in later section

However, the first and foremost challenge is to determine the *number of layers* - the depth and the *number of neurons* in each layer - the width of the model. Every neural network will require a single input layer and a single output layer. The number of neurons in input layer and output layer are also easy to determine based on the number of variables in the data processed. The challenge remains for how to design hidden layers. Artificial neural network only requires hidden layers if the data must be separated non-linearly. First proposed in 1989, Cybenko [45] stated theorem of universal approximation ability for a single hidden layer neural network for sigmoid activation function. The theorem stated that a feed forward network containing one hidden layer with finite number of neurons can approximate continuous functions with mild assumptions on the activation function. Hornik later in 1991 [46] expanded the work of Cybenko that the multilayer architecture itself of the neural network gives it the potential of being universal approximations.

Theorem 5 (Universal Approximation Theorem). *Fix a continuous function* $\sigma : \mathbb{R} \to \mathbb{R}$ *(activation function) and positive integers* d, D. *The function* σ *is not a polynomial if and only if, for every continuous function* $f : \mathbb{R}^d \to \mathbb{R}^D$ *(target function), every compact subset* K *of* $\mathbb{R}^d$, *and every* $\epsilon > 0$ *there exists a continuous function* $f_\epsilon : \mathbb{R}^d \to \mathbb{R}^D$ *(the layer output) with representation*

$$f_\epsilon = W_2 \circ \sigma \circ W_1,$$

where W_2, W_1 *are composable affine maps and* $\circ$ *denotes component-wise composition, such that the approximation bound*

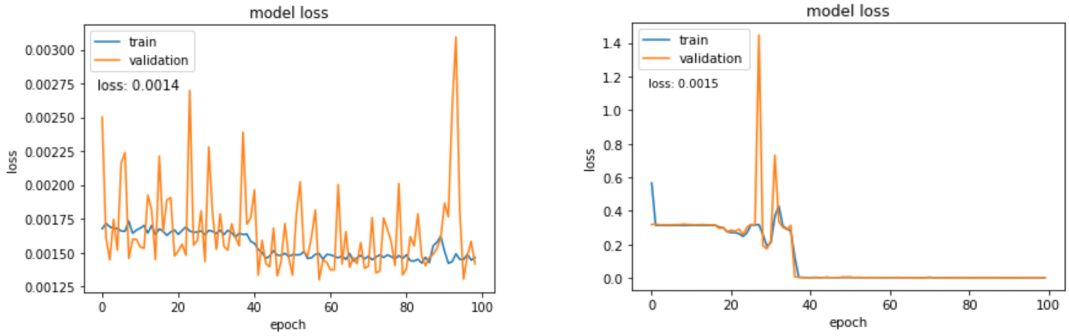
$$sup_{x \in K} ||f(x) - f_\epsilon(x)|| < \epsilon$$

holds for any ϵ *arbitrarily small (distance from* f *to* f_ϵ *can be infinitely small).*

However, although this theorem proves that a single hidden layer network can learn anything, it does not show how easy it can learn. That is why nowadays the multilayer perceptron becomes more and more popular. Hinton from his research in 2006 [47] contributed a lot to learning algorithms for deep neural networks with multiple hidden layers. Empirical practice shows that, one or two hidden layers network can be good to solve simple tasks, many hidden layers can be fruitful for more complex tasks, image recognition problems. However, the more hidden layers does not assure the better performance of the neural network model. While too few layers and neurons can make the model *underfitting*, too many parameters can lead to an *overfitting* model. [48]

Next, we demonstrate some results from empirical experiences to see how the loss value can vary while changing number of layers and number of neurons in each layer.

Decision of these two numbers is even more important when layer type is classic *dense layer*, indicating that each neuron of the preceding layer will be fully connected to every neuron of the next layer. Besides dense layers, modern neural networks also have dropout, convolutional, pooling or recurrent layers. Type of layer is also one of hyper-parameter that the creater should consider before construct a network since each layer type has its own purpose and can raise the efficiency if being used correctly. In this thesis work, we will only use dense layer because of the regression property of the problem and with the purpose of *reversing network* from outputs tracing back to the inputs.



(a) Model of 4 layers, 4 neurons each

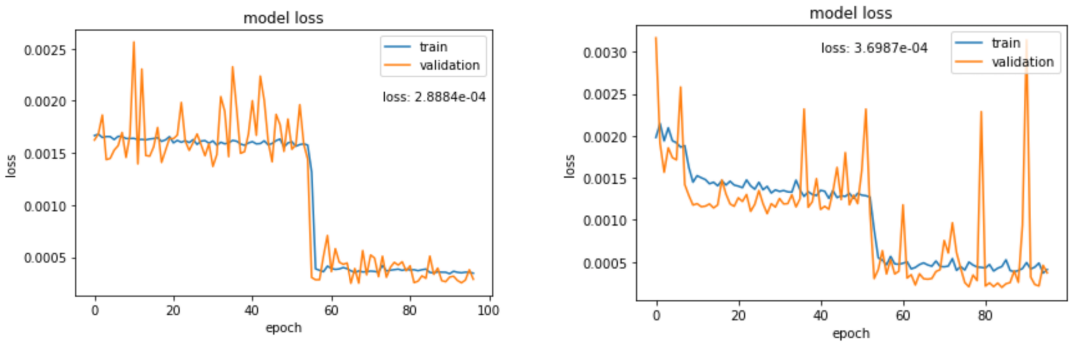
(b) Model of 10 layers, 4 neurons each

Figure 6: Training loss changed with various choices of the number of layers [2]

In what follows we will use *learning curves*, see more [50], to illustrate the loss value during the training of the same data set with two model architectures. In figure 6, the first diagram describes model with only 4 layers and the second is ones having 10 layers, including input and output layers. In both models, each layer contains 4 neurons, that means, we only expand *the depth* but not *the width* of models. The fact shows that even we increase four times the number of layers, general loss after training 100 epochs seems to remain the same. Furthermore, the second try with 10 layers seems *unstable* when around epoch 30 loss value of validation set suddenly climbed up to 1.4. It can be explained by the huge number of trainable weight parameters which may cause *heavy* computing then explode some numbers during computation. Although in this case $e-02$ is not a bad result, a considerable improvement can be made with altering number of neurons in layers. Figure 7 (a) shows model of also 4 layers but having 8 neurons instead of 4 in the second hidden layer. Its training loss is $2.9e-04$ and even can be better with network of 6 layers including two hidden layers with 8 neurons. At the end of training, we also see some signals of blowing values and all through the progress validation loss highly fluctuates around model loss. Generally

[2]Loss value in all figures in this section is measured by mean square error

in this case we can raise the number of training epochs to observe the moving trend.



(a) Model of 4 layers, neurons 4-4-8-4

(b) Model of 6 layers, neurons 4-4-8-8-4-4

Figure 7: Loss with changing number of neurons in each layer

One more remark for figures 6 and 7 is that we excluded loss values of from one to three first training epochs. Because of the stochastic initialization - choosing randomly the first step to begin gradient descent, loss value of the several first epochs often considerably higher than the next ones. For example, in the case of model in figure 6 - (a), loss value of the first epoch is 1.0158 which is 1000 times as much as 0.0016 of the second epoch. We more want to focus on how the loss varies during training process to investigate the trend or any uncommon exploding or vanishing values so the first loss value epoch is not necessary.

There are several more hyper-parameters such as activation function, network weight initialization, momentum, optimizer or learning rate which will be presented in the next subsection. Minimizing the loss function is the original target of training neural network and this target can be achieved by expanding the depth or the width of network or changing other control model hyper-parameters. However, reducing the loss function value may need to trade off by complex computation progress or much longer training time because of increase in number of parameters. Ultimately, the selection of model architecture would come down to trial and error and what our goals are.

5.4 Optimizers and Learning rate

An optimizer of a training model is an optimization method or a strategy used to control various attributes of the network to reduce loss in an efficient way. **Learning rate** is one of the key hyper-parameters in a deep training model. In general, deep learning neural networks are used the stochastic gradient descent algorithm ([42] - [43]) or some other extension of this method to train model. Therefore, we will demonstrate the term learning rate with respect to the stochastic gradient descent method.

Consider the pair combining by the input x an the corresponding output y: (x_i, y_i). The loss function l measure the distance from the prediction $h(x, w)$ to the actual value y: $Q(w) = l(h(x, w), y)$ or the loss on each pair of data $Q_i(w) = l(h(x_i, w), y_i)$. Over the training set $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, the *empirical risk* is measured by

$$E_n(h) = \frac{1}{n} \sum_{i=1}^{n} l(h(x_i, w), y_i) = \frac{1}{n} \sum_{i=1}^{n} Q_i(w), \tag{30}$$

while $E(h)$ denotes the *expected risk* measuring the general performance - the expected performance on the future data sets. The statistical learning theory [44] showed that instead of minimizing the expected risk, it is enough to just minimize the empirical risk $E_n(h)$.

Then now it worth to focus on the minimizing problem of an object function:

$$Q(w) = \frac{1}{n} \sum_{i=1}^{n} Q_i(n), \tag{31}$$

where the weight vector w is the parameter need to estimate to minimize $Q(w)$.

Gradient descent is an iterative algorithm method, each iteration updates the weight w based on the gradient of $E_n(h)$:

$$w_{t+1} := w_t - \eta \nabla Q(w_t) = w_t - \frac{\eta}{n} \sum_{i=1}^{n} \nabla Q_i(w_t), \tag{32}$$

here η is a chosen gain. Under some assumptions, when the learning rate is small enough and the initial estimate w_t is closed enough with the optimum, the iterative algorithm can archives linear convergence, that is, $\log \rho \sim t$ where ρ is residual error.

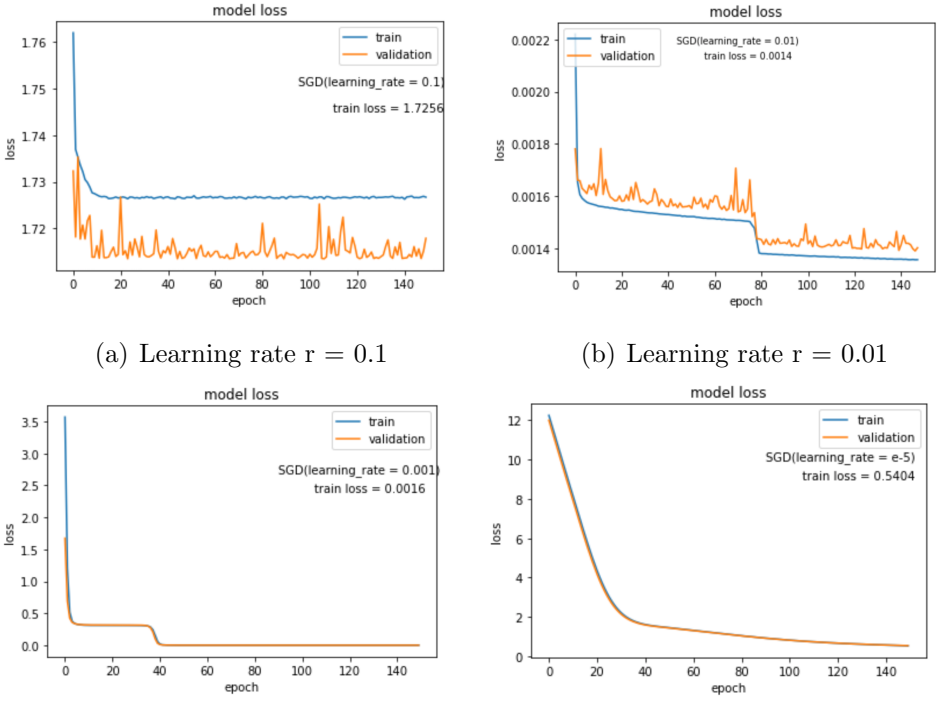
Stochastic gradient descent is a simplification of gradient descent algorithm. Each iteration will no longer compute exactly the gradient of $E_n(h)$, instead pick a random pair (x_t, y_t) from training set:

$$w_{t+1} := w_t - \eta \nabla Q_t(w_t). \tag{33}$$

By this way, stochastic gradient descent method does not require as much memory as gradient descent since it does not need to store gradient of the whole dataset. On the other hand, stochastic sample choosing may cause high variance in parameters or keep recomputing even after achieving global minimum.

Therefore, in short, stochastic gradient descent is an optimization that uses the error gradient of the current state to update the weights of the model for the next stage. The amount that the weights updated through each stage is call step size, in this case step size is equal to learning rate times gradient. Used in neural network training, the learning rate often has range between 0.0 to 1.0. If a learning rate is too

large, the algorithm may converge too quickly to a sub-optimal solution but can not reach the optimum because of a huge jump thereby missing it. A small learning rate will require more training epochs and cause stuck sometimes if it is too small.



(a) Learning rate r = 0.1

(b) Learning rate r = 0.01

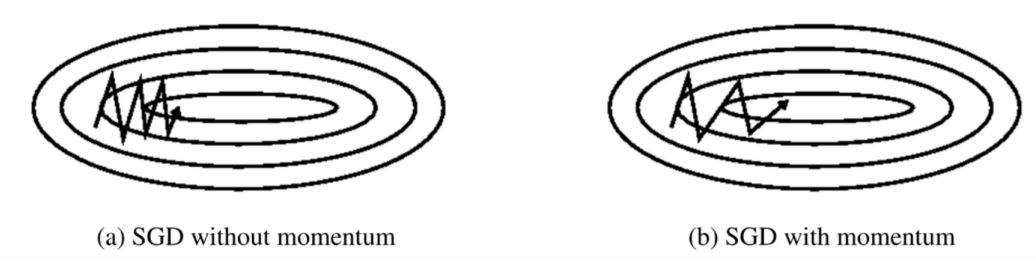
(c) Learning rate r = 0.001

(d) Learning rate r = 10^{-5}

Figure 8: Comparison between various learning rates (SGD Optimizer)

Plotting can be a good way to see how impact of learning rate on training loss. Figure 8 illustrates models trained using Adam optimizer with different learning rates, said $0.1, 10^{-2}, 10^{-3}, 10^{-5}$ respectively. By observation, networks with learning rate 0.01 and 0.001 seem to have acceptable training loss and show the potential of convergence. Train with rate 10^{-5} also seems to be gradually convergent however it requires more epochs to train until when we can get desirable loss value. While with larger learning rate ($r = 0.1$) training loss is 1000 times larger and train loss always higher than validation loss, that means the network is currently underfitting [50].

One extension of stochastic gradient descent is to combine it with momentum method.

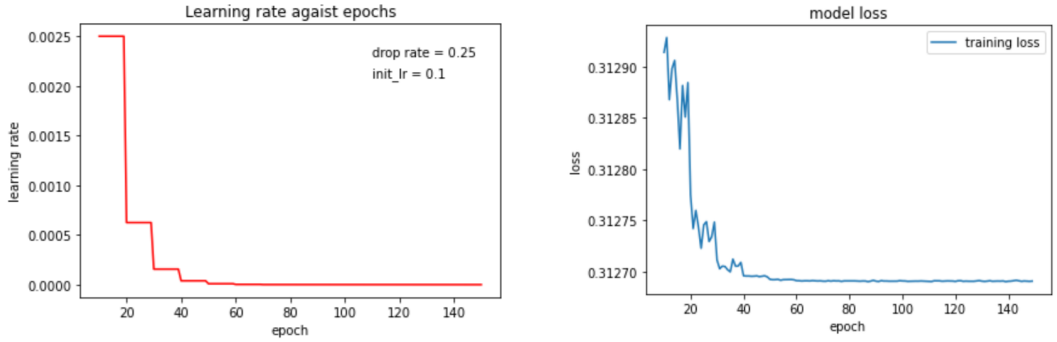


(a) SGD without momentum (b) SGD with momentum

Figure 9: SGD method without and with using momentum [3]

When using stochastic gradient descent with momentum, the momentum term tends to speed up the convergence to local minima after each epoch like the ball will roll faster and faster down the hill. The time to convergence can be improved since we do not need to oscillate too much up and down the y-axis anymore. More about momentum will be presented later when we mention Adam method.

It is general that with relatively high learning rate the loss function jump down rapidly at first but smaller learning rate shows better trend to converge and ability to find optimum point. The idea then is that we want to start with a high learning rate then gradually decrease its value during training. One popular form is **decay learning rate** which reduces the initial learning rate by a number of percentage after some training epochs. The percentage and the set of epochs after that the value of learning rate changed are not necessary constants. With Keras, users can establish *learning rate schedule* to define a function updating learning rate with a specific rule.



(a) Learning rate vs Epoch (b) Training loss vs Epoch

Figure 10: Using Decay Learning Rate with Stochastic Gradient Descent method

The model shown in Figure 10 has training loss around 0.3, much better than value 1.7256 we have when train model with fixed learning rate 0.1 as in Figure 8 (a). However, it is not always true that decay learning rate is better than fixed learning rate.

[3]Source: Genevieve B. Orr

Other well-known form of learning rate is **cyclical learning rate** proposed by Leslie Smith in [51]. In this form, the learning rate will vary between upper and lower bounds. There are three update rules presented by the author in the paper: triangular update rule, triangular update schedule with fixed decay and with exponential decay, among those triangular update rule is the most popular. The rule to update triangular cyclical learning rate (CLR) is simple to understand

$$lr = lr_{min} + (lr_{max} - lr_{min})(\max(0, 1 - x)), \tag{34}$$

defined

$$x = \left| \frac{epochCounter}{stepsize} - 2cycle + 1 \right|$$

and

$$cycle = \left\lfloor 1 + \frac{epochCounter}{2stepsize} \right\rfloor,$$

where lr_{min} and lr_{max} are two boundary values, $epochCounter$ is the current training epoch and 2 $stepsize$ is equal to a $cycle$. As be seen in the formula, at the beginning learning rate starts at the base vale lr_{min}, reaches maximum lr_{max} at the half of cycle then goes down and meets the base again.

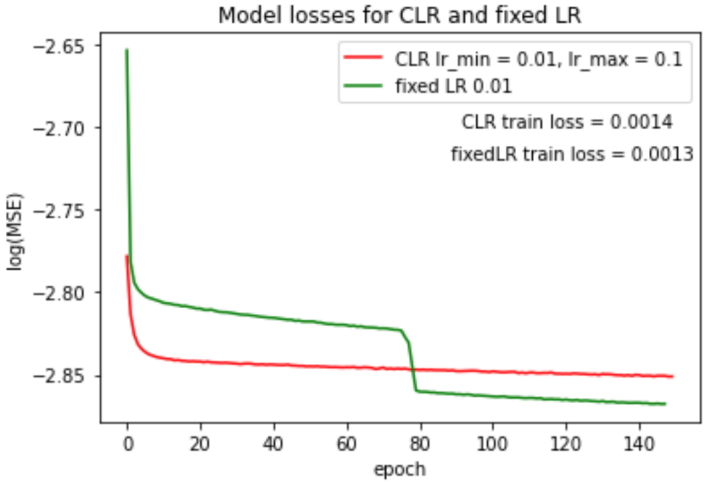


Figure 11: Model loss using fixed learning rate and cyclical learning rate (SGD optimizer)

Figure 11 shows an example when we tested the cyclical learning rate to stochastic gradient descent optimizer in our model. We can see that although the cyclical learning rate showed advantage at first, eventually it could not perform a better overall performance in comparison with the fixed learning rate usage. This can be happened in reality for many reasons. Thus, again, building model is a process of trials and false.

A much more useful way to update the learning rate is to consider it as a variable of the model and adjust it based on some parameters during training, called **adaptive learning rate**. Three methods **AdaGrad**, **RMSProp** and **Adam** provided in

Keras allow us to implement this idea, where Adam [49] combines benefits of both two preceded methods with the Stochastic Gradient Descent with momentum. Overview of various optimization algorithms can be seen at [52]. Here we will only show some short notes of these algorithms.

Adagrad is Adaptive Gradient Algorithm. It performs larger updates for infrequent parameters and smaller updates for frequent parameters so it is best suited with sparse data in large scale. Formula for weight updating in Adagrad is

$$w_{t+1} = w_t - \eta \frac{g_t}{\sqrt{G_t + \epsilon}}, \tag{35}$$

where G_t is the sum of the squares of the past gradient, g_t is the gradient, η and ϵ are chosen gains.

RMSProp is Root Mean Square Propagation, in which the learning rate is divided by the average of the exponential decay of squared gradients.

$$w_{t+1} = w_t - \eta \frac{g_t}{\sqrt{(1-\gamma)g_{t-1}^2 + \gamma g_t + \epsilon}}, \tag{36}$$

where γ is the decay term having value range of $[0, 1]$.

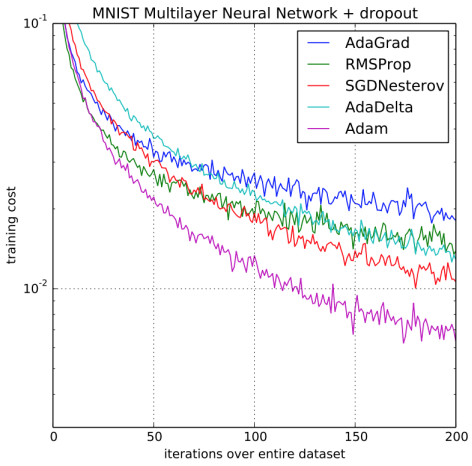


Figure 12: Comparison between various algorithms for MNIST images [49]

The name **Adam** is derived from adaptive moment estimation. The steps that Adam uses to update weight parameters of model is presented as follow. This algorithm uses the first and second moment estimators of gradient to adapt the learning rate. The first momentum of gradient is computed by exponential moving average

$$\mathbb{E}(m_t) = \mathbb{E}(g_t)$$
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t,$$

where β_1 is by default equal to 0.9, $m_0 = 0$ and g is gradient of the current epoch. Using the recursive formula we can get $m_t = (1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i$ and

$$\begin{aligned}
\mathbb{E}(m_t) &= \mathbb{E}(g_t) \\
&= \mathbb{E}((1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i) \\
&= \mathbb{E}(g_i)(1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} + \zeta \\
&= \mathbb{E}(g_i)(1 - \beta_1^t) + \zeta,
\end{aligned}$$

as taking an approximation for $\mathbb{E}(g_i)$ then we can pull it out the sum and then need to add an error ζ. Bias corrected estimators for the first momentum will be

$$\hat{m_t} = \frac{m_t}{1 - \beta_1}. \tag{37}$$

We process proceed similarly for the second moment of gradient, start with

$$\begin{aligned}
\mathbb{E}(v_t) &= \mathbb{E}(g_t) \\
v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2,
\end{aligned}$$

where $\beta_2 = 0.999$, $v_0 = 0$. Then we get bias correction for the second momentum

$$\hat{v_t} = \frac{v_t}{1 - \beta_2}. \tag{38}$$

The last step is to use estimators (37) and (38) in updating model parameters. The way it is done in Adam is simple

$$w_t = w_{t-1} - \eta \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon}, \tag{39}$$

where similar to equation of weight update rule in stochastic gradient descent (33), w is weight parameter, η is a chosen gain also known as the default learning rate and ϵ is a small term (usually 10^{-8}) preventing division by zero.

6 On the inversion of Neural networks

6.1 Invertible neural networks

In all of the above sections we are talking about how to predict a quantities directly from given input parameters. However, in reality, when analyzing complex systems, there are common problems that the parameters of interest can not be computed directly. A mapping that measures quantities $\mathbf{y}$ from the hidden parameters $\mathbf{x}$ is called the *forward process*. The *inverse process* can understand simply as tracing back $\mathbf{y} \rightarrow \mathbf{x}$ for free after well trained forward process. From a mathematical perspective, invertible architectures enable several unique guarantees:

- guaranteed preservation of mutual information and exact access to invariants of deep networks [61].
- memory-saving gradient computation [62].
- fast analytical invertibility [63].

By [60], invertible neural networks are characterized by properties:

- the mapping from inputs to outputs is bijective, i.e. its inverse exists,
- both forward and inverse mapping are efficiently computable, and
- both mappings have a tractable Jacobian, which allows explicit computation of posterior probabilities.

Modeling the conditional posterior of an inverse process is a classical statistical problem that, in theory, may be handled using Bayesian approaches. Unfortunately, the inverse problem by Bayesian method is often intractable since some important information has been lost during the forward process. Various methods has been established for a more efficient alternative, we, however, will not go into further details.

6.2 Mathematical inversion

As state at the beginning, implied volatility is an important parameter for traders and investors that over the years, volatility forecasting becomes a hot research topics on finance. However, in this work I want to mention a *mathematical inversion* of neural network. Basically, mathematical inversion is like normal inversion but instead of building train model, in the backward direction, we use mathematics, which may help to reduce unnecessary complexity if applying invertible neural networks to a simpler problem.

From the formula of computing output at each layer in the forward training network (28)

$$x^{(j)} = f\left(W_j x^{(j-1)} + b_j\right).$$

Now the target is to trace back the input values $x^{(j-1)}$ when we have all information of outputs $x^{(j)}$, weight W_j and bias parameters b_j, activation functions f. Assume that we can do it in the sense that all function and weight matrices are invertible, we can find $x^{(j-1)}$ by

$$x^{(j-1)} = W_j^{-1}\left(f^{-1}(x^{(j)}) - b_j\right), \tag{40}$$

where the exponent (-1) presents the inversion.

However, for the first reverse step, instead of taking model prediction as initial input of (40) we use the true value of y in the training set.

$$x^{(m)} = W_m^{-1}\left(f^{-1}(y) - b_m\right), \tag{41}$$

where the exponent (-1) presents the inversion and m is the number of hidden layers in neural network, so in that case $x^{(m)}$ presents the hidden layer right before the output layer.

Inverse of activation functions

First and foremost condition for a neural network being invertible is that it uses all invertible activation functions. We know that not every function can be invertible. Let f be a function of domain X and its codomain is Y, to have an inverse, each element in Y must correspond to no more than one element in X, said injection. If f^{-1} is a function on the set Y, each element $y \in Y$ corresponds to some $x \in X$ then f^{-1} is called surjection. To be invertible, a function must be both injection and surjection. The following are few inversions of often used activation functions:

- Inverse function of Sigmoid activation function is the Logit function $f^{-1}(x) = \frac{1}{a}\ln\left(\frac{x}{1-x}\right)$.
- RELU does not have inverse function since the RELU function itself is not an injection.
- Inversion of LeakyRELU is $f^{-1}(x) = \begin{cases} \frac{1}{\alpha}x & x < 0 \\ x & x \geq 0 \end{cases}$

However, we are lucky since we can control the choices of activation functions at the beginning when ones builds the network architecture. All problem now leaves for inversion of the weight matrices parameters.

Inverse of weight matrices

In linear algebra, an $n \times n$ square matrix $\mathbf{A}$ is called invertible if there exists an $n \times n$ square matrix $\mathbf{B}$ such that

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n,$$

where $\mathbf{I}_n$ denotes the identity matrix and the multiplication used is the ordinary matrix multiplication. If this is the case, then the matrix $\mathbf{A}$ determines uniquely the matrix $\mathbf{B}$ which is also called the inverse of $\mathbf{A}$ and denoted by $\mathbf{A}^{-1}$.

A requirement for a square matrix to have an inverse is the determinant of it must not be zero. A square matrix having an inverse is called non-singular. In the other hand, a matrix does not have an inverse is called singular matrix. Over the field of real numbers, the set of singular $n \times n$ matrices is a null set, that means has Lebesgue measure zero. Indeed, singular matrices are the roots of the determinant function. The determinant is a polynomial in the entries of the matrix. By Richard Caron from University of Windsor [64]

Theorem 6. *A polynomial function on $\mathbb{R}^n$ to $\mathbb{R}$, is either identically 0, or non-zero almost everywhere.*

Proof. We will prove the theorem by induction on n.

Denote n-dimensional Lebesgue measure by λ_n.

If $n = 1$ and p is polynomial, not the zero polynomial, of degree m then p has at most m roots so $\lambda_1 \{x : p(x) = 0\} = 0$. Suppose the result is true for polynomial with $n - 1$ variables.

Since p is continuous then the zero set of p denoted by $\mathbf{Z}(p)$ is a measurable subset of $\mathbb{R}^n$.

Now we present a non-trivial polynomial p with n variables, degree m in x_n

$$p(\mathbf{x}, x_n) = \sum_{j=0}^{m} p_j(\mathbf{x}) x_n^j$$

where $\mathbf{x} = (x_1, x_2, ..., x_n)$ and $p_0, p_1, ..., p_m$ are polynomials in $n - 1$ variables and at least p_m is non-trivial.

Let $(\mathbf{x}, x_n)$ such that $p(\mathbf{x}, x_n) = 0$ then or $p_0(\mathbf{x}) = p_1(\mathbf{x}) = ... = p_m(\mathbf{x}) = 0$ or x_n is a root of polynomial $p_{\mathbf{x}}(z) = \sum_{j=0}^{m} p_j(\mathbf{x}) z^j$.

Let $\mathbf{A}$ and $\mathbf{B}$ be the subset of $\mathbb{R}^n$ where the above conditions hold respectively, then $\mathbf{Z}(p) = \mathbf{A} \cup \mathbf{B}$

Recall that $\mathbf{x}$ is $n-1$ vector then by the inductive hypothesis, it is true that $\mathbf{A}$ has measure zero.

For a fixed $\mathbf{x}$, the set $\{z : p_{\mathbf{x}}(z) = 0\}$ is finite by the fundamental theorem of algebra. Since a finite set has measure zero in $\mathbb{R}$ then we can conclude that $\mathbf{B}$ has measure zero. ■

Thus in the language of measure theory, almost all $n \times n$ matrices are invertible. In practice, we still have possibility to encounter a singular square matrix or matrices which are invertible but close to a non-invertible matrix, which may cause problems. In some sense, it can be said that we will be *more lucky* to reach our target of reversing neural network if we construct a network with all weight parameters being square matrices. That means in a fully connected network architecture, all number of neurons in input, hidden and output layers should be the same.

Nevertheless, as presented in the previous section, model of neural network can show much better performance with a small change in number of neurons in layers. We, therefore, want to test the case that reversing model where weight matrices are not square.

Moore [65] seems to have introduced the notion of an inverse of a singular matrix in 1920. Some extension but no systematic analysis of the topic was made until 1955, when Penrose, ignorant of the earlier work, redefined the Moore inverse in a more or less different way [66].

Definition - Pseudoinverse for real matrices. For $\mathbf{A} \in \mathbb{R}^{m \times n}$, a **pseudoinverse** of $\mathbf{A}$ is a matrix $\mathbf{A}^+ \in \mathbb{R}^{n \times m}$ if satisfies all the following Moore-Penrose conditions:

1. $\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$
2. $\mathbf{A}^+\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+$
3. $(\mathbf{A}\mathbf{A}^+)^\top = \mathbf{A}\mathbf{A}^+$
4. $(\mathbf{A}^+\mathbf{A})^\top = \mathbf{A}^+\mathbf{A}$

In fact, $\mathbf{A}^+$ exists for any matrix $\mathbf{A}$. But when $\mathbf{A}$ has full rank then the inverse matrix can represented as left or right inverse. If matrix $\mathbf{A}$ $m \times n$ has rank n $(n \leq m)$ then $\mathbf{A}$ has a left inverse, that is $n \times m$ matrix $\mathbf{A}^+ = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top$ such that $\mathbf{A}^+\mathbf{A} = \mathbf{I}_n$. And if $\mathbf{A}$ has rank m $(m \leq n)$ then it has a right inverse, that is $\mathbf{A}\mathbf{A}^+ = \mathbf{I}_n$ and $\mathbf{A}^+ = \mathbf{A}^\top(\mathbf{A}\mathbf{A}^\top)^{-1}$. If the inverse matrix of $\mathbf{A}$ exists, then the pseudoinverse will coincide with it.

There are several methods to compute the pseudoinverse. However, we will mention here the most famous ones is to use the singular value decomposition, which also is the method used in programming languages. If $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top$ is the singular value

decomposition of $\mathbf{A}$, then $\mathbf{A}^+ = \mathbf{V}\Sigma^+\mathbf{U}^\top$ where Σ is a diagonal matrix, $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices. To get pseudoinverse of Σ, we take the reciprocal of each non-zero element on the diagonal then transpose the matrix.

The pseudoinverse can give best approximation solution to any system of linear equations in the sense of least squares. For a given system of equations

$$\mathbf{A}x = b,$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$. Using Moore-Penrose pseudoinverse will lead to an approximate solution $z = \mathbf{A}^+b$ such that $\forall x \in \mathbb{R}^n$, $||\mathbf{A}x - b|| \geq ||\mathbf{A}z - b||$ where $||.||$ denotes Euclidean norm.

In a short discussion about Moore-Penrose inverse and application to solving linear system [67], a group of authors stated conclusions may be useful in our cases. In the case *over-determined linear system*, that means there are more equations than unknowns (or simply $n < m$), we can always find a unique solution using the ingenious approach of least squares. However, in the case of *under-determined linear system*, more unknowns than equations ($m < n$), the solution is not unique and the system may have $(n - m)$ non-trivial independent solutions. Therefore, if an under-determined system remains under-determined even in the least square approach, then the solution derived with the help of Moore-Penrose pseudoinverse can just be considered as *a natural solution* but not the unique ones.

So in general, it is not always possible to find the desired input when using the inversion of rectangle matrices. Especially in the case feeding layer dimension is less than its of receiving layer (the weight matrix is under-determined), there may exist infinite solutions that satisfy formula, pseudoinverse can give one solution with least squares approach but have no information to compute exactly or nearly the input that we have at the origin.

7 Designing the models

7.1 Data sets

As a data-driven approach, the quality of a data set has an impact on the performance of the resulting model. Focusing on the European call options, which price can be calculated by the Equation (22) so we first generate a random data of 100,000 samples having 6 features $\{S, K, \sigma, T, r, C\}$.

Parameters	Range	Type
Stock price S	[99, 201]	integer
Strike price K	[19,51]	integer
Maturity T (in year)	[1,5]	integer
Risk free rate r	[0, 0.05]	float
Volatility σ	[0,1]	float
Call option price C	[49, 185]	integer

Table 3: Dataset 1 including 6 variables $\{S, K, \sigma, T, r, C\}$

The first impression can be seen from this dataset is that the ranges of stock prices and option prices, between 50 to 200, are much higher than the ranges of risk free rate and volatility which are always below 1.

Method	Train loss	Output parameter differneces (mean, variance)				
		K	S	T	r	C
Adam	0.9411	(-0.18, 3e-2)	(-0.7, 4.5e-2)	(0.34, 0.59)	(-0.03, 2.3e-4)	(0.3, 3.3)

Table 4: Table of mean and variance of output differences after training with Dataset 1

In order to test how this dataset can work, we set the input includes 5 variables $\{\sigma, K, S, T, r\}$ and output has also 5 dimensions $\{C, K, S, T, r\}$ and used a training model partially similarly to Model 1, which will be described in the upcoming subsection. In fact, after some training attempts, models result in some undesirable properties. For instance, when using optimizer Adam or RMSProp, outputs after training always present a negative risk free rate, which can be unrealistic sometimes and we want to eliminate this bad prediction. The reason may partially come from too large distances between ranges of parameters then when training, it is hard to find good weight parameters that can work well with both small range values and large range

values. During complicated computation, some values can be blown up or vanished or be negative as we have. The negative problem no longer happens when the model uses SGD optimization. However, this algorithm can not reach optimal situations, training loss is always high since there is a high variance in Call option price difference between model predicted value and the true value of training data (see Table 4).

Based on properties of features in this specific dataset, it is reasonable to think about re-scaling data. In detail, instead of Stock, Strike and Option price our data will contain only ratio of prices in comparison with Strike price K described in Table 5 below. This dataset is also the one used to test hyper-parameters in the previous section.

Parameters	Train Set Range	Type
Stock price (S/K)	[1.98, 10.53]	float
Maturity T (in year)	[1,5]	integer
Risk free rate r	[0, 0.05]	float
Volatility σ	[0,1]	float
Call option price (C/K)	[0.98, 9.72]	float

Table 5: Dataset 2 including 5 variables $\{S/K, \sigma, T, r, C/K\}$

Depending on the change of data, the input and output of neural network also need to be changed to 4 dimension, $\{\sigma, S/K, T, r\}$ plays the role of the input and $\{C/K, S/K, T, r\}$ the output.

Parameters	Train Set	Wide Test Set	Narrow Test Set
Stock price (S/K)	(0.7, 3.4)	(0.6, 4.3)	(0.9, 2.8)
Maturity T (in year)	[2,5]	[1,6]	[3,4]
Risk free rate r	[0.01, 0.05]	[0.0, 0.07]	[0.02, 0.04]
Volatility σ	[0.05, 0.9]	[0.01, 1]	[0.1, 0.8]
Call option price (C/K)	(0.0, 2.8)	(0.0, 3.7)	(0.07, 2.0)

Table 6: Training and Testing Datasets

Because of the randomness while generating data, we can see some unrealistic points in Dataset 2, for example, the ratio between stock price and strike price being up to over 10 is too big. To get rid of these unwanted features in the final data, we change ranges of value in price and generate data of 100,000 samples for only the training set (presented in Table 6). Besides the training data set, we also generate two testing

data sets of size 10,000 each, one with features in the same ranges as training data and one with slightly wider ranges to see how the model deals with it since empirical experiences show that neural networks seem to work less efficient near the border of the training set and to be able to forecast precisely with a centered data.

7.2 Model selections and performances

In this part, we present three neural network architectures, as well as hyper-parameter selections based on purpose. Each of the three models is trained using the specified training datasets listed in the preceding subsection and validated using both a larger and a smaller testing range.

Parameters	Model 1	Model 2
Layers	3	3
Neurons	4 - 4 - 4	4 - 8 - 4
Activation function	LeakyRELU	LeakyRELU
Optimizer	Adam	Adam
Loss function	MSE	MSE
Drop-out rate	0	0
Batch-normalization	No	No
Kernel Initializer	Random uniform	Random uniform

Table 7: Model selections

Model 1 and **Model 2** were designed to assess the potential of mathematically reversing the trained neural network, which is the main purpose of this study. Since the regression form of the model is used, almost all of the hyper-parameters of both models are similar. Nevertheless, the architecture of neurons differs. Although Model 1 has the same number of neurons in each layer, Model 2 has a significantly different number of neurons in hidden layers, requiring the use of Moore-Penrose pseudoinverse. By training and testing, we realize an interesting feature being true for both models is that it seems like an increase in the number of layers in a model does not mean an increase in model performance. Both models can work well with only 3 layers, adding more hidden layers is redundant because of time consuming and the same or even worse training loss. As this fact is mentioned in the previous chapter when discussing hyper-parameters of neural networks, we do not need to have more hidden layers to get good predictions and for regression problem networks can find a good convergence with a few layers. However, we also know that this is not always the case.

The loss function in used for updating parameters of network is the **mean square error (MSE)**.

$$\textbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Otherwise, we use other functions to test model efficiency, such as root mean square error (RMSE), mean absolute error (MAE), and mean absolute percentage error (MAPE).

$$\textbf{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$\textbf{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$\textbf{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|,$$

where y_i stands for observations and $\hat{y}_i$ are predictions.

Table 8 illustrates the results in the sense of training loss MSE and other metrics after forward training Model 1 and Model 2. It succeeds to show that changing a little in neurons architecture helps slightly to increase performance and both models prove its incorrectness when predict data point near the boundary, shown in the performance of the testing-wide set.

Dataset		**MSE**	**RMSE**	**MAE**	**MAPE**
Training	Model 1	3.9638×10^{-4}	0.0199	0.0085	1.3961
	Model 2	5.4659×10^{-5}	0.0074	0.0031	0.7116
Testing-wide	Model 1	0.0389	0.1972	0.0377	1142.35
	Model 2	0.0027	0.0516	0.0155	617.00
Testing-narrow	Model 1	1.5562×10^{-4}	0.0125	0.0058	0.7814
	Model 2	1.9224×10^{-5}	0.0044	0.0021	0.3636

Table 8: Forward training results with Model 1 and Model 2

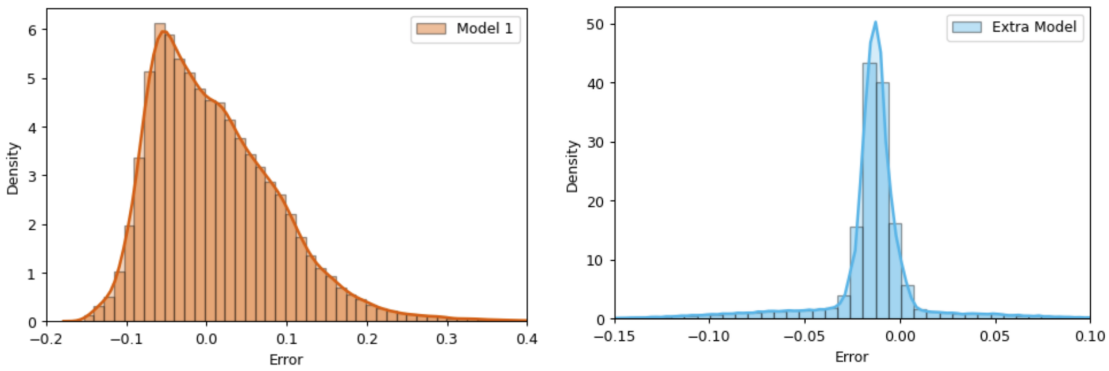
On purpose of comparing the precision of training with reverse models and the standard forward approach, we created a **IV-ANN model** that takes all other variables as input and the only output is the corresponding volatility value. This kind of model had been tested in and brought in very optimistic results. At least we hope we can at some extent compare between the accuracy of IV-ANN model output and traceback output of Model 1. Since if we are lucky, the trained Model 1 can be mathematically invertible then the general function matching input and output can be considered as an

bijection, thus, we can easily do backward the process with a little acceptable loss. In that case, the quality of the backward computation depends a lot on how accurate the forward training can do. In general, the selection of hyper-parameters for the IV-ANN model is the same with Model 1 and Model 2 except for the number of layers and number of neurons in each layer. An initial choice for the IV-ANN model gives us the below result, in Table 9. Note that these results are not the best performance that we can achieve because of many reasons such as the efficiency of computers, the choices in ranges of data sets or the network architecture itself. Good results for applying neural networks to compute financial volatility can be found in further supporting references with the hybrid with other models and techniques. Here our purpose is to implement an example model used for initial comparison the efficiency of mathematical reverse method.

Dataset	**MSE**	**RMSE**	**MAE**	**MAPE**
Training	9.3484×10^{-4}	0.0306	0.0206	9.3966
Testing-wide	0.0052	0.0723	0.0440	34.5253
Testing-narrow	7.4781×10^{-4}	0.0273	0.0187	6.8604

Table 9: Training result with IV-ANN Model

However, we can not yet compare the training losses in Table 8 and Table 9 because of the difference in output dimension. In order to assess how well each model forecasts volatility values, we need to look at the volatility error distribution output of each model, especially IV-ANN model and Model 1.



(a) Model 1 (b) IV-ANN model

Figure 13: Error density performance on the Training Dataset

For IV-ANN model, differences between $\hat{\sigma}$ the prediction and σ the true value given in dataset are considered. To take error of volatility from Model 1 and Model 2 we need

to do the mathematical backward process according to formula (40) and formula (41) getting $\hat{\sigma}$ then take $\hat{\sigma}$ to minus σ. The IV-ANN model gives us the difference mean about -0.0147 and 0.00072 variance, mean of absolute difference is 0.0206 while Model 1 having corresponding difference mean, difference variance and absolute difference mean are 0.0111, 0.00622 and 0.0619. As expected, Model 2 because of doing Moore-Penrose pseudoinverse can not recompute correctly the input, results in difference mean of 4.085 and variance up to 7.237.

Figure 13 plots the error histogram and density curve performance of two models IV-ANN and Model 1 with respect to the training data set. The peak of volatility error after backward computing is around 0.06-0.07 while its peak of only forward training model IV-ANN is 0.02-0.03. As can be seen, Model 1 produces a much wider range of error than IV-ANN model. It is understandable since the IV-ANN model training only one variable that is volatility then the error can be more concentrated in comparison with the Model 1 which requires output of 4 dimensions.

7.3 In comparison with Newton-Raphson method

We compare the performance of Model 1, IV-ANN model and Newton-Raphson method in terms of accuracy in predicting the implied volatility. We define a precision error ϵ to evaluate accuracy of methods. A sample is considered as being approximated correctly if

$$|\hat{\sigma} - \sigma| \leq \epsilon.$$

Depending on the statistical parameters mean absolute difference of IV-ANN model and Model 1 (in the Table 10 and Figure 14 the name Model 1 is replaced by Reversed model to emphasize on the method of mathematical reverse the neural network), which are around 0.02 and 0.06 respectively, we can set 3 values for epsilon ϵ: 0.01, 0.02 and 0.06.

	Newton-Raphson method	**IV-ANN model**	**Reversed model**
$\epsilon = 0.01$	76.173 %	71.616 %	9.175 %
$\epsilon = 0.02$	77.45 %	87.085 %	18.656 %
$\epsilon = 0.06$	81.653 %	95.932 %	56.847 %

Table 10: Accuracy of predicting implied volatility on training dataset

As result shown in Table 10, in this case, predicting implied volatility by feed forward neural network has slightly better performance than Newton iteration method.

In this specified experience, we limited the number if iterations in Newton method down to 10 iterations. Therefore, the accuracy of Newton - Raphson method can reach much higher level in reality practice. Forecasting by usage of mathematically reversed network has low accuracy. It does, however, indicate a potential future improvement of the method. It proves that first, the reverse can be done and second, gives an acceptable error rate on some loose accuracy criteria. One superior advantage of using neural network in general rather than iteration method is the less requirement on CPU and GPU usage since it only requires matrix multiplication or inner product [53]. We hope somehow we can raise the accuracy percentage by adjusting to a more efficient network architecture.

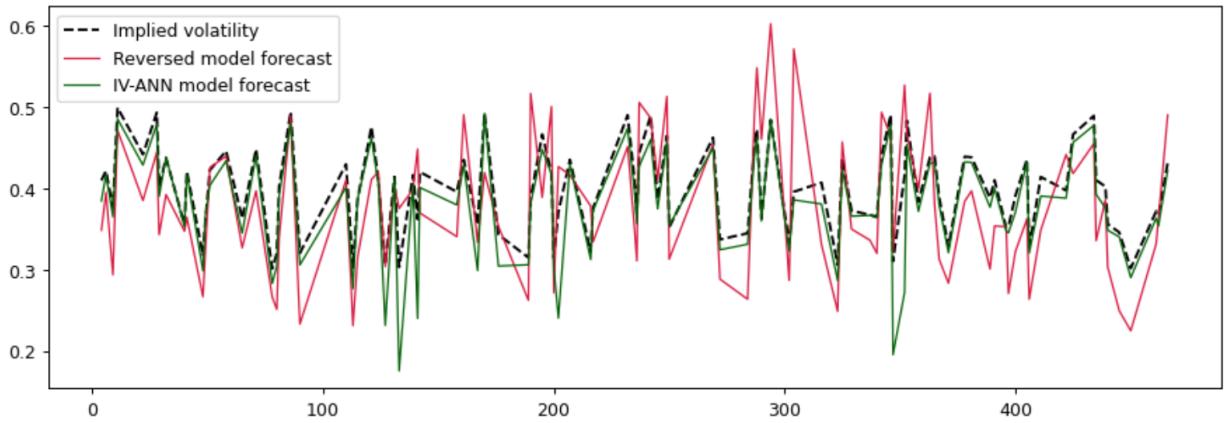


Figure 14: Graph of the actual implied volatility and the neural network forecast of implied volatility

8 Conclusion

In this thesis study we have proposed a brand new approach to compute implied volatility supported by the foremost famous Black-Scholes model combined with deep neural network technique. Besides, literature of other traditional most used methods to forecast implied volatility along with their pros and cons are discussed. We realize some unchangeable drawbacks of Newton numerical iteration methods are often eliminated by applying deep learning into action, particularly using deep neural networks to train models of high dimensions. Self-organizing learning and least square methods are used leading to potential high accuracy and more importantly, to resolve problems of zero derivatives and optimize the usage of CPU and GPU caused by Newton method. Concerning the very fact that finding implied volatility is indeed finding the root of the inverse Black-Scholes formula and in some cases the invertible neural network can predict with high accuracy, we implement a replacement approach that uses literature of inverse functions and inverse matrices to mathematically reverse a trained neural network architecture. We considered two cases of mathematical inverse problem, the first case requires invertible activation functions and all square trainable weight matrices while the second case has looser criteria, does not need square matrices. We initially developed two models to assess the feasibility of this new approach, and the first of them showed promise. We tested the forecasting accuracy of this promising reversed model with other traditional forward networks and Newton method. Our initially numerical result showed that despite that the new approach did not present a superior performance to other methods, its accuracy can be acceptable in the sense of loose criteria.

Future research

Since the topic of research is relatively new and we only did one trial model in this thesis paper, there is much work that can be done to improve this topic in the future.

a) In the aspect of testing the feasibility of mathematically reversed neural networks:

We can implement this approach to other financial models rather than Black-Scholes model. One potential example is the Heston model, which uses the stochastic implied volatility instead of constant volatility as in Black-Scholes formula. In fact, the stochastic volatility shows a superior performance in practical in comparison with a constant volatility. We will be happy if the inverse neural network can work well with this parameter.

Furthermore, we can extend this study for other financial parameters rather than implied volatility.

b) In the aspect of improving the existing application with Black - Scholes model:

Concerning the relationship between the forward training and the backward performance, we hope that improvement on feed forward training can bring about a better reversed approximation. The way people process input variables may affect the accuracy of the forecast neural network like the way we have changed from Dataset1 to Dataset2. Input data can be improved when focusing on the difference of the time value and intrinsic value option price, the way authors in [53] did to enhance their ANN model loss.

Improvement may be also possible through experimentation with other data and network architectures. Since the network in use has four input dimensions and four output dimensions while only one feature of volatility is really concerned, it is obvious to think about an architecture that reduce the not the overall loss but only the specified needed dimension error.

9 Acknowledgments

References

[1] Robert Brown F.R.S. Hon. M.R.S.E. and R.I. Acad. V.P.L.S. XXVII. A brief account of microscopical observations made in the months of June, July and August 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies. *The Philosophical Magazine* 1827; 4:21, 161-173.

[2] Peter Mörters and Yuval Peres Brownian Motion - Draft version of May 25, 2008.

[3] Samuelson, P.A. Proof That Properly Anticipated Prices Fluctuate Randomly. *Industrial Management Review* 1965; 6, 41-49.

[4] Fama, E. Efficient Capital Markets: A Review of Theory and Empirical Work. *Journal of Finance* 1970; 25, 383-417.

[5] Fama, E. Efficient Capital Markets: II. *Journal of Finance* 1991; 46, 1575-1617.

[6] Shaheen Borna and Dheeraj Sharma How much trust should risk managers place on "Brownian motions" of financial markets? *International Journal of Emerging Markets* 2011; Vol.6, Issue 1.

[7] George Chalamandaris and A. G. Malliaris Chapter Itô's Calculus and the Derivation of the Black–Scholes Option-Pricing Model. *Handbook of Quantitative Finance and Risk Management - Volume 1* pp. 447-470.

[8] Brenner, M., Subrahmanyam, M.G. A simple formula to compute the implied standard deviation. *Finan. Analysts J.* 1988; 44(5), pp. 80–83.

[9] Bharadia, M.A., N. Christofides, and G.R. Salkin. Computing the Black-Scholes Implied. Volatility. *Advances in Futures and Options Research* 1996; pp. 15-29.

[10] Don. M. Chance A Generalized Simple Formula to Compute the Implied Volatility. *Financial Review.* 1996; 31(4), pp. 859–867.

[11] Li, Steven A New Formula for Computing Implied Volatility. *Applied Mathematics and Computation.* 2005; 170(1), pp. 611-625.

[12] Donald R. Chambers An Improved Approach to Computing Implied Volatility. *The Financial Review.* 2001; 38, pp. 89-100.

[13] Steven Manaster, Gary Koehler The Calculation of Implied Variances from the Black-Scholes Model: A Note. *The Journal of Finance.* 1982; Vol 37. No 1.

[14] SGiuseppe Orlando, Giovanni Taglialatela A review on implied volatility calculation. *Journal of Computational and Applied Mathematics.* 2017; Vol. 320; pp. 202-220.

[15] Hull, J. C. Options, Futures and Other Derivatives, 7th Edition. *New Jersey: Prentice Hall* 2009.

[16] Hull, J. and A. White The pricing of Options on Assets with Stochastic Volatilities. *Journal of Finance* 1987; 42, 281-300.

[17] Gikhman, I., and A. V. Skorokhod Investment to the Theory of Random Processes. *Saunders* 1969; 387-389

[18] Arnold, L. Stochastic Differential Equation: Theory and Applications. *John Wiley & Sons* 1974; 90-99.

[19] Baxter, M. and A. Rennie Financial Calculus. *Cambridge: University Press* 1996.

[20] Wilmott, P. Paul Wilmott on Quantitative Finance. *John Wiley & Sons* 2000.

[21] Xisheng Yu and Xiaoke Xie On Derivations if Black-Scholes Greek Letters. *Research Journal of Finance and Accounting* 2013; Vol.4, No.6.

[22] Chen, H.Y., Lee, C.F. and Shi, W.K. Derivations and applications of Greek letters - review and integration. *Handbook of Quantitative Finance and Risk Management, Part III* 2010; 491-503.

[23] Black, F. and Scholes, M. The pricing of options and corporate liabilities. *Journal of Political Economy* 1973; Vol.81, 637-659.

[24] Merton, R. C. The theory of rational option pricing. *Bell Journal of Economics and Management Science* 1973; Vol.4, pp. 141-183.

[25] Bachelier, L. Theory of Speculation (translation of 1900 French edition). *The Random Character of Stock Market Prices, MIT Press* 1964; pp. 17-78.

[26] Stephen J. Taylor Modeling stochastic volatility: A review and comparative study. *Mathematical Finance* 1994; Vol.4, Issue 2, pp. 183-204.

[27] Stefan Gerhold Can There Be An Explicit Formula for Implied Volatility?. *Applied Mathematics E-Notes.* 2013; pp. 17-24.

[28] L. Lipshitz D-finite power series. *J. Algebra.* 1989; pp. 353-373.

[29] R. P. Stanley Differentiably finite power series. *European J. Combin.* 1980; pp. 175-188.

[30] Krznaric, Matthew J. Comparison of Option Price from Black-Scholes Model to Actual Values. *Honors Research Projects* 2016; 396.

[31] Espen G Haug and Nassim N Taleb. Option traders use (very) sophisticated heuristics, never the Black-Scholes-Merton formula. *Journal of Economic Behavior and Organization* 2011; 77(2): pp. 97-106.

[32] Léon Bottou, Frank E. Curtis, Jorge Nocedal Optimization Methods for Large-Scale Machine Learning. *arXiv:1606.04838 [stat.ML]* 2018.

[33] Martin T. Hagan, Howard B. Demuth, Mark Hudson Beale, Orlando De Jesus Neural Network Design (Electrical Engineering); 2nd edition, 2014.

[34] W. McCulloch and W. Pitts A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics.* 1943; Vol. 5, pp. 115–133.

[35] D. O. Hebb The Organization of Behavior. *New York: Wiley.* 1949.

[36] F. Rosenblatt The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review.* 1958; Vol. 65, pp. 386–408.

[37] J. A. Anderson A simple neural network generating an interactive memory. *Mathematical Biosciences.* 1972; Vol. 14, pp. 197–220.

[38] T. Kohonen Correlation matrix memories. *IEEE Transactions on Computers.* 1972; Vol. 21, pp. 353–359.

[39] S. Grossberg Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. *Biological Cybernetics.* 1976; Vol. 23, pp. 121–134.

[40] J. J. Hopfield Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences.* 1982; Vol.79, pp. 2554–2558.

[41] D. E. Rumelhart and J. L. McClelland Parallel Distributed Processing: Explorations in the Microstructure of Cognition. *MA: MIT Press.* 1986; Vol. 1, Cambridge.

[42] Ketkar N. Stochastic Gradient Descent. In: Deep Learning with Python *Apress, Berkeley, CA.* 2017.

[43] Bottou L. Large-Scale Machine Learning with Stochastic Gradient Descent. *Proceedings of COMPSTAT'2010* 2010; pp 177-186.

[44] V. N. Vapnik An overview of statistical learning theory. *IEEE Transactions on Neural Networks.* Sept. 1999; Vol. 10, no. 5, pp. 988-999.

[45] Cybenko, G. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems.* 1989; 2(4), pp. 303-314.

[46] Kurt Hornik Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks.* 1991; 4(2), pp. 251-257.

[47] Hinton, G. E.; Osindero, S.; Teh, Y. W. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation.* 2006; 18(7), pp. 1527-1554.

[48] XavierGuyon, Jian-fengYao On the Underfitting and Overfitting Sets of Models Chosen by Order Selection Criteria. *Journal of Multivariate Analysis.* August 1999; Volume 70, Issue 2, pp. 221-249.

[49] Diederik P. Kingma, Jimmy Lei Ba ADAM: A Method For Stochastic Optimization. *Published as a conference paper at ICLR.* 2015.

[50] Michel Jose Anzanello, Flavio Sanson Fogliatto Learning curve models and applications: Literature review and research directions. *International Journal of Industrial Ergonomics.* September 2011. Volume 41, Issue 5, pp. 573-583.

[51] Leslie N. Smith Cyclical Learning Rates for Training Neural Networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)* 2017.

[52] Sebastian Ruder An overview of gradient descent optimization algorithms. *arXiv:1609.04747* Sept. 2016.

[53] Shuaiqiang Liu, Cornelis W. Oosterlee and Sander M. Bohte Pricing Options and Computing Implied Volatilities using Neural Networks. *Risks* 2019; 7(1), 16.

[54] Mary Malliaris, Linda M. Salchenberger Using neural networks to forecast the S&P 100 implied volatility. *Neurocomputing.* 1996; pp. 183-195.

[55] F. Gonzalez Miranda & N. Burgess Modelling market volatilities: the neural network perspective. *The European Journal of Finance.* 1997; Vol. 3; Issue 2.

[56] S. D. Bekiros, D. A. Georgoutsos Direction-of-change forecasting using a volatility-based recurrent neural network. *Journal of Forecasting.* 2008; Vol. 27; Issue 5; pp. 407-417.

[57] LIN Yan, YANG Jianhui Option Pricing Model Based on Newton-Raphson Iteration and RBF Neural Network Using Implied Volatility. *Canada Social Science.* 2016; Vol.12; No.8; pp. 25-29.

[58] Werner Kristjanpoller, Anton Fadic, Marcel C. Minutolo Volatility forecast using hybrid Neural Network models. *Expert Systems with Applications.* 2014; Vol.41; Issue 5; pp. 2437-2442.

[59] Isaac Faber Comparing Historical and Implied Volatility Estimates in Efficient Portfolios. *Journal of Finacne and Investment Analysis.* 2013; Vol. 2; No. 4; pp. 57-82.

[60] Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W. Pellegrini, Ralf S. Klessen, Lena Maier-Hein, Carsten Rother, Ullrich Köthe Analyzing Inverse Problems with Invertible Neural Networks. *Published as a conference paper at ICLR 2019.* 2019.

[61] Jorn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-Revnet: Deep invertible networks. *International Conference on Learning Representations.* 2018.

[62] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *In Advances in Neural Information Processing Systems* 2017; pp. 2214–2224.

[63] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516* 2014.

[64] Richard Caron The Zero Set of a Polynomial. *Technical Report* May 2005.

[65] Moore, E. H. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society.* 1920; 26 (9): 394–95.

[66] Penrose, Roger A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society.* 1955; 51 (3): 406–13.

[67] J. López-Bonilla, R. López-Vázquez, S. Vidal-Beltrán Moore-Penrose's inverse and solutions of linear systems. *World Scientific News.* 2018.

[68] Peter Jackel By Implication. *Wilmott 26.* July 2006; p. 60-66.

[69] Peter Jackel Let's be Rational. *Wilmott.* March 2015; p. 40-53.

[70] Minqiang Li & Kyuseok Lee An adaptive successive over-relaxation method for computing the Black-Scholes implied volatility. *Quantitative Finance, Taylor & Francis Journals.* 2011; Vol. 11(8); pp. 1245-1269.