University of Economics in Prague

Faculty of Finance and Accounting

Department of Banking and Insurance

Study programme: Finance and Accountancy

Field of study: Financial Engineering

Option Pricing Using Machine Learning

MASTER THESIS

Author: Bc. Peter Pagáč

Final thesis supervisor: prof. RNDr. Jiří Witzany, Ph.D.

2018

Declaration

I declare that I carried out this diploma thesis independently, and only with the parencited sources, literature and other professional sources.

In Prague on 31.05.2018 .....

Author signature

Acknowledgments

I would first like to thank my thesis advisor prof. RNDr. Jiří Witzany, Ph.D.. The door to Prof. Witzany office was always open whenever I ran into a trouble spot or had a question about my research or writing.

I must also express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Abstract

The thesis focuses on option pricing using neural networks. The goal is to use state-of-the-art recurrent neural network for option pricing and comparing it's performance to Black-Scholes model and another neural network architecture. The reader is in theoretical part, provided with foundations of Black-Scholes model and machine learning. We also introduce and describe in great detail feed-forward networks and recurrent neural networks with Long short-term memory units. Subsequently, in the practical part we present our data and describe training of neural networks. Furthermore, we present our proposed approach for pricing options using LSTM networks. In addition, we present detailed technical aspects and software use for training. The last part is dedicated to results obtained by all models. In conclusion, we suggest wide range of improvements and ideas for further research.

Keywords

Neural networks, Option pricing, Long short-term memory

Abstrakt

Práce se zaměřuje na oceňování opcí pomocí neuronových sítí. Cílem je využívat nejmodernější rekurentní neuronovou síť pro stanovení ceny za opce a porovnat její výkon s modelem Black-Scholes a další architekturou neuronové sítě. Čtenář je v teoretické části vybaven základy modelu Black-Scholes a strojového učení. Dále uvádíme a podrobně popisujeme feed-forward neuronové sítě a rekurentní neuronovou síť s Long short-term memory jednotkami. Následně v praktické části uvádíme použité data a popisujeme trénink neuronových sítí. Dále uvádíme náš navrhovaný přístup k oceňování opcí pomocí sítí LSTM. Navíc uvádíme podrobné technické aspekty a využití softwaru pro modelování. Poslední část je věnována výsledkům získaným všemi modely. V závěru mimo jiné navrhujeme velký počet vylepšení a nápadů pro další výzkum.

Klíčová slova

Neuronové sítě, Oceňování opcí, Long short-term memory

Contents

Introduction

There are various instruments to trade on financial markets in addition to direct investments in equities, bonds, currencies or commodities, it is possible to use financial derivatives. Financial derivatives derive it's value from an underlying asset which can be essentially everything from equity to cryptocurrencies. Bank of International Settlement estimates that nominal outstanding amount of derivatives contracts was in 2017 more than 542439 billions of USD (International Settlement 2018) thus market for derivatives is huge and important. When trading derivatives, it is crucial that derivatives are correctly priced. Correct pricing may allow to use derivatives by more institutions with less capital requirements which in turn may allow to lend more money for smaller interests. One of very popular derivatives are options. These are popular as a hedging tool against price swing, but also for speculating purposes. That is why we choose them.

In recent years, we have seen huge improvement in computing power and computer possibilities which is turn allowed to use computationally hungry algorithms for real life purposes. Neural networks are one example. Neural network are successfully used in areas such as face recognition, autonomous driving of vehicles, hand writing recognition and others. There is no doubt that neural networks can be successfully used in finance. Pricing of derivatives can be one area where using neural networks can be beneficial. Given that price of an underlying asset is difficult to forecast and thus it is sometimes difficult to price the risk associated. Neural network however are capable of learning very complex rules. It may be possible that when some time series is showed to neural network, it may find such patterns that where not found by human or traditional tools so far which in turn may lead to more accurate forecasting and pricing. And that is going to be main theme of our thesis, are neural network capable to price real life options more precisely that traditional methods?

Our goal is to show that it is possible to use neural networks for option pricing and to to compare traditional methods. We assume that they may yield more accurate results. Concretely, we would like to use special king of neural networks with mem-

ory to exploit time series nature of option price. Even though that neural networks where first used for option pricing in early 90s, we haven't found any previously published papers that would use network with memory presented in this thesis for option pricing. Moreover, majority of authors use recurrent networks for forecasting using only univariate time series. We propose a technique of how to transform and fit neural networks to multivariate time series representing data about options. We believe that technique we present is interesting and it's possible to use it in real life situations. Our side motivation is to present an example of neural networks in world of finance. Some people might think that they are too complicated for using in common situations. So to refute this, we present libraries and frameworks we used for working with neural networks that are very easy to use. All software we use is open-source and free to use. The thesis is divided into five chapters. In the first one, we study and present traditional method for option pricing, Black-Scholes model. In the next chapter we introduce machine learning as a field. We focus on neural networks which we explain in great detail. In the third chapter, we present our proposed method and all models we had used. In the forth chapter, we present data we used. The last chapter is dedicated to results presentation.

Chapter 1

Options and Their Pricing

1.1 Options

An option is a financial derivative that represents a contract between a buyer (a holder of the option) and a seller. There are two types of options. A call option gives the buyer a right to buy an underlying asset at a certain price (a strike price or an exercise price) on a certain date (an expiration date or a maturity). A put option gives the buyer a right to sell the underlying asset or an instrument at a certain price on a certain date. The date depends on the type of the option. European options may only be exercised on the expiration date. American options may be exercised at any time on or prior the expiration date. The underlying asset might be an equity, a bond, a future, an index, a commodity, and many others. The other side of a contact is the seller side. The seller has an obligation to full-fill his side of the transaction if the buyer decides to exercise his right. This means to buy or sell the underlying asset for the exercise price apart from the actual spot price of the underlying asset. An option can also be described by it's moneyness. That is how close is it's strike price to a spot price of an underlying asset. An option can be at-the-money (ATM), the option's strike price is roughly equal to the spot price. In-the-money (ITM), the option's strike price is lower the spot price given call option or the option's strike price is higher than the spot price given put option. And out-of-the-money (OTM), the option's strike price is higher the spot price given call option or the option's strike price is lower then the spot price given put option. It is important to remember that the buyer has the right, but not an obligation to buy or sell the underlying asset. This property distinguishes options from forward and future contracts, where the buyer is required to buy or sell the underlying asset. On the other hand, it is free

of costs to enter future or forward contract,but it costs something to buy an option. This cost is called *the premium*. The premium is paid by the buyer to the seller for writing the option and the seller keeps the premium even if the buyer decides not to exercise his option. We will refer to the premium as a price of an option or vice verse in this thesis, but we assume that both mean the cost of acquiring the option.

1.2 Option Premium

The premium usually represents, in case of equity options, a price for a single underlying share. In case of index options, an option contract has a multiplier that determines the overall price of the contract. Usually the options are traded which means that the premium is negotiated on the market and it's subject of market powers. Thus the price premium is derived by supply and demand. Other aspects of an option such as strike price, maturity or type of an option are specified in the contract and will not usually change during a life of the option.

In theory, the option premium consists of two parts: **intrinsic value** and **time value**. Intrinsic value is a value that holder would get if the option was exercised immediately. It basically represents difference between actual spot price of the underlying asset and the strike price. It can be represented as follows:
for call options

$$max(S - K; 0)$$

and for put options

$$max(K - S; 0).$$

Time value is the value that option price exceeds the intrinsic value of the option. It represents amount that the option seller is demanding for taking risk that the underlying value would change so much that the option would end up in the money. It is positively correlated with the maturity. The time value however is subject to exponential decay.

While the intrinsic value is given by above formulas, the time value is affected by many different factors. In addition, many of them are not quantifiable so it is necessary to proceed in an approximation manner. One example of successfully approximations is the Black-Scholes model which is introduced in following sections. The factors that affect the option price are the following variables:

- Spot price of the underlying asset. If the spot price of the underlying asset increases, the price of a call option will also rise. For put options, increase of

the underlying asset reduces the value of the option.

- Time to maturity. It's effect is ambiguous. For European options, the effect depends on the relationship between spot and forward prices i. e. if the marked is normal or inverted. A call option value increases with longer time to maturity if the spot prices are expected to grow. However if the spot prices are expected to decrease than the value could decrease as-well.

- Strike price. The option price typically increase when the option is more ITM and decreases when the option is more OTM thus for call options increasing strike price decrease option price and for puts increase the price.

- Interest rate. The increasing domestic interest rate has positive effect on the underlying asset value which in turn has positive effect on a call option value and negative on put option value.

- Volatility. Option price is increasing function of option price *ceteris paribus*. The logic behind this is that if the volatility is high than potential gain is also high, but the loss is fixed.

1.3 Put-call parity

The term **put-call parity** refers to the relationship between option premiums of corresponding call and put options (with same underlying asset, same strike price and same maturity). The relationship is used for example in situation when we have option value of a call option and we want to calculate value of the put option or vice-versa.

Put-call parity for European options:

$$C_t - P_t = S_t - Ke^{-r(T-t)}$$

This can be derived using two portfolios. The portfolio A which consists of European put option with an underlying stock and one share of the stock and the portfolio B which consists of European call option with the underlying stock and cash amount of $Ke^{-r(T-t)}$. In a given time T, both portfolios must have the same value $max(S_t, K)$. And because both options are European, it must hold that in time t for both portfolios $P_t + S_t = C_t + Ke^{-r(T-t)}$ which is the same as the above equation.

1.4 Bounds for Option Prices

We define bounds for option prices that would be used in practical part for filtering data. For a European call option price on a stock, we define lower bound in the absence of arbitrage opportunities as follows:

$$p \geq max(0; S_0 - Ke^{-rT})$$

Assume that we consider following portfolios:

- Portfolio A: one European call option and cash amount of Ke^{-rt}
- Portfolio B: one share

In portfolio A, if the cash was invested for risk-free rate than it will grove to K in time T. If $S_T > K$ than the call option would be exercised and portfolio A would be worth S_T. If however $S_T < K$ than the option would not be exercised and the portfolio would be worth K. So at time T the portfolio is worth $max(S_T, K)$. Portfolio B is worth S_T at time T. So if portfolio A is always worth at least S_T and could be more at maturity, than $c \geq S_0 - Ke^{-rT}$. And because options cannot have negative value than $c \geq 0$ and hence the above condition.

For European put option price on a stock we define lower bound as follows in the absence of arbitrage opportunities:

$$p \geq max(Ke^{-rT} - S_0; 0)$$

If we consider following portfolios:

- Portfolio A: one European put option and one share
- Portfolio B: cash amount of Ke^{-rT}

In portfolio A, if $S_T > K$ than the option is exercised on maturity and the portfolio is worth K. If however $S_T > K$ than the option is not exercised and the portfolio is worth S_T. So at time T, the portfolio is worth $max(S_T, L)$. In portfolio B is worth K in time T. So if portfolio A is always worth at least S_T and could be more at maturity, than $p \geq Ke^{-rT} - S_0$. And because the option cannot have negative value than $p \geq 0$ and hence the above condition.

1.5 Black-Scholes-Merton Model

Options has become very popular and important derivative in the financial world. It's popularity has increased significantly after invention of pricing formula known as **Black-Scholes** or Black-Scholes-Merton formula by Black and Scholes (Black and Scholes 1973) and Merton (Merton 1973) in 1973. The formula is a part of a model of a financial market which contains at-least one risky and one risk-less asset. There are other relatively strict assumptions of this model:

- The price of the stock is guided by the geometric Brownian motion with constant drift and volatility.
- The model assumes a risk-free rate r that is constant and same for all instruments with due maturity.
- The model assumes that returns are log-normally distributed and markets are efficient.
- The underlying stock do not pay out dividends.
- There is no possibility of arbitrage.
- Trading of the stock is continuous.
- There are no transaction costs or fees and it is possible to buy or sell any proportion of the stock.
- Short selling is allowed with full use of proceeds.

It is however possible to relax some of these assumptions so it fits more real world conditions.

1.5.1 Itô's Lemma

The Black-Scholes model is based on assumption that time series of an option price with an underlying asset is a function of time to expiration and a spot price of the underlying asset. In general, we can say that any derivative is a function of stochastic variables price and time. Important part in understanding of such function is **Itô's Lemma** (Itô 1950). If a variable follows Itô process of:

$$dx = a(x,t)dt + b(x,t)dz, \tag{1.1}$$

where dz is a Wiener process and a and b are functions of x and time t. Variance of x is b^2 and drift rate is a. Itô's lemma states that a function G of x and t follows Itô process of

$$dG = (\frac{\partial G}{\partial x}a + \frac{\partial G}{\partial t} + \frac{1}{2}\frac{\partial^2 G}{\partial x^2}b^2)dt + \frac{\partial G}{\partial x}bdz, \tag{1.2}$$

with a drift rate:

$$\frac{\partial G}{\partial x}a + \frac{\partial G}{\partial t} + \frac{1}{2}\frac{\partial^2 G}{\partial x^2}b^2$$

and variance:

$$\frac{\partial G}{\partial x}b^2$$

1.5.2 Derivation of the Black-Scholes Differential Equation

Let's assume that the underlying asset follows a geometric Brownian motion:

$$dS = \mu S dt + \sigma S dz, \tag{1.3}$$

where dz is a Wiener process. Next suppose that the price of the option is f with underlying price of the stock S. The price f must be than function of S and time t. We use Itô's lemma 1.5.1 and show that if 1.3 is a process of underlying stock price than function of df of S and t is than:

$$df = (\frac{\partial f}{\partial S}\mu S + \frac{\partial f}{\partial t} + \frac{1}{2}\frac{\partial^2}{S^2})dt + \frac{\partial f}{\partial S}\sigma S dz \tag{1.4}$$

Now we choose portfolio[1] consisting of one short option and $\frac{\partial f}{\partial S}$ shares of the stock. The value of the portfolio Π is:

$$\Pi = -f + \frac{\partial f}{\partial S}S \tag{1.5}$$

We can define $d\Pi$ as a change of value of the portfolio in the time interval $\triangle t$:

$$\triangle\Pi = -\triangle f + \frac{\partial f}{\partial S}\triangle S \tag{1.6}$$

Substituting discrete versions of equations 1.3 and 1.4 in equation 1.6 we get:

$$\triangle\Pi = (-\frac{\partial f}{\partial S}\mu S - \frac{\partial f}{\partial t} - \frac{1}{2}\frac{\partial^2 f}{S^2})\triangle t - \frac{\partial f}{\partial S}\sigma\triangle z + \frac{\partial f}{\partial S}(\mu S\triangle t + \sigma S\triangle z). \tag{1.7}$$

This can be adjusted to:

$$\triangle\Pi = (-\frac{\partial f}{\partial t} - \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\sigma^2 S^2)\triangle t. \tag{1.8}$$

[1] delta hedge-portfolio

The equation is missing $\triangle z$ which was the only source of randomness. This means that if the no-arbitrage condition holds than the portfolio must be risk-less and it's profit must be r which is a rate of return from short-term risk-free security. It follows:

$$\triangle \Pi = r\Pi \triangle t. \tag{1.9}$$

It is possible to substitute equations 1.5 and 1.8 into 1.6 so we get:

$$(\frac{\partial f}{\partial t} + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\sigma^2 S^2)\triangle t = r(f - \frac{\partial f}{\partial S}S)\triangle t \tag{1.10}$$

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} = rf \tag{1.11}$$

Equation 1.11 is the Black-Scholes-Merton Differential Equation. It has many solutions. We get the particular solution equation for the price of a particular derivative using the right boundary conditions. The conditions are similar the conditions defined in 1.4 except for discounting K which is omitted. Two points should be emphasised. The first is that the portfolio 1.5 is not risk-less permanently, but only infinitesimally short period of time. It is given by fact that if S and t change than $\frac{\partial f}{\partial S}$ also changes. One would need to adjust proportion of the stock and derivative in portfolio in order to keep the portfolio risk-less. The second thing is an observation that equation 1.11 does not contain any variable that would regard a risk-preference of an investor. This allows to price an option for any level of risk preference.

1.5.3 Black-Scholes Formula

Prices for European put and call options calculates by the Black-Scholes formula. Price the call option:

$$c_0 = S_0 N(d_1) - Ke^{-rT}N(d_2), \tag{1.12}$$

where,

$$d_1 = \frac{\ln(\frac{S_0}{K}) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}, \tag{1.13}$$

$$d_2 = \frac{\ln(\frac{S_0}{K}) + (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}, \tag{1.14}$$

and $N(x)$ is the standard normal distribution cumulative probability function, σ is a volatility or a standard deviation of the stock's returns.

And for the put option with payoff given by $f = max(K - S, 0)$:

$$p_0 = Ke^{-rT}N(-d_2) - S_0 N(-d_1), \tag{1.15}$$

where d_1 is 1.13 and d_2 is 1.14.

The formula was derived in original paper (Black and Scholes 1973) setting up and solving 1.11 (Black–Scholes PDE). Although solving the Black–Scholes PDE is a bit difficult it holds for many other different derivatives and only the boundary conditions makes the difference applicable to an European options. If the Black–Scholes PDE has not analytic solution, than it is possible to solve it using numerical methods. In next section, we will prove the call option pricing formula using risk-neutral pricing.

1.5.4 Proof of Black-Scholes Formula for Call Option

We take 1.3 to a new probability measure Q. This means that the drift rate μ with respect to Q is the risk-free rate r, thus:

$$dS = rSdt + \sigma Sdz \tag{1.16}$$

Now this means that S_T has with respect to Q following log-normal distribution:

$$\ln S_T \sim N(\ln S_0 + (r - \frac{1}{2}\sigma^2)T, \sigma^2 T). \tag{1.17}$$

Now we take log-normally distributed variable $S = S_T$ with density $g(S)$:

$$\ln S \sim N(m, w^2), \tag{1.18}$$

where

$$m = \ln S_0 + (r - \frac{1}{2}\sigma^2)T, \tag{1.19}$$

and

$$w^2 = \sigma^2 T \tag{1.20}$$

The formula will be verified by evaluation of:

$$E[max(S - K, 0)] = \int_K^\infty (S - K)g(S)dS. \tag{1.21}$$

Now we can transform S into the standardised normal variable X by $X = \frac{\ln S - m}{w}$ and use density function of X: $\varphi(X) = \frac{1}{\sqrt{2\pi}} e^{-\frac{X^2}{2}}$. Probability of $g(S)dS$ must be equal to $\varphi(X)dX$ and thus:

$$\begin{aligned} E[max(S - K, 0)] &= \int_{\frac{\ln K - m}{w}}^\infty (e^{Xw+m} - K)\varphi(X)dX = \\ &= \int_{\frac{\ln K - m}{w}}^\infty \frac{1}{\sqrt{2\pi}} e^{\frac{(-X^2+2Xw+2m)}{2}} dX - K \int_{\frac{\ln K - m}{w}}^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{-X^2}{2}} dX \end{aligned} \tag{1.22}$$

It is optimal to evaluate both integers. To evaluate the first one, we need to complete the square in the exponent: $\frac{-X^2+2Xw+2m}{2} = \frac{-(X-w)^2+2m+w^2}{2}$. To evaluate the second one, we use that $\varphi(-X) = \varphi(X)$ and $\int_x \infty \varphi(X)dX = \int_{-\infty} -x\varphi(X)dX = N(-x)$. So,

$$\int_{\frac{\ln K-m}{w}}^{\infty} \frac{1}{\sqrt{2\pi}} e^{\frac{-X^2+2Xw+2m}{2}} dX = e^{\frac{m+w^2}{2}} int_{\frac{\ln K-m}{w}}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{(X+2)^2}{2}} dX = \\ e^{\frac{m+w^2}{2}} N(w-(\ln K-m)/w) \tag{1.23}$$

Now using 1.18, 1.19 and 1.20 we can show that:

$$\frac{w-(\ln K-m)}{w} = \frac{-(\ln K-m)+w^2}{w} = \frac{\frac{-\ln K+\ln S_0+rT-\sigma^2T}{2+\sigma^2T}}{\sigma\sqrt{T}} = \frac{\frac{\ln S_0}{K}+\frac{(r+\sigma^2T)}{2}}{\sigma\sqrt{T}} = d_1, \\ \frac{-\ln K-m}{w} = \frac{\frac{\ln S_0}{K}+\frac{(r-\sigma^2T)}{2}\sigma\sqrt{T}}{=} d_2, \tag{1.24}$$

and

$$e^{\frac{m+w^2}{2}} = e^{\ln S_0+rT} = S_0e^{rT} \tag{1.25}$$

And finally using $f_0 = e^{-rT}E_q[f_T]$ we get the formula[2]:

$$c = e^{-rT}(S_0e^{rT}N(d_1) - KN(d_2)) = S_0N(d_1) - e^{-rT}KN(d_2) \tag{1.26}$$

[2]Formula for put option can be verified the same way

Chapter 2

Machine Learning

2.1 Machine Learning

The term *machine learning* was first used by Arthur Samuel in his paper Computer Games from 1959 (L. Samuel 2000). He used machine-learning procedures to verify that computer is capable to learn how to play game of checkers better than the person who programmed it. This mean that the program is not simple set of rules and play book scenarios programmed by the programmer but the program is able to learn new knowledge about the game and apply it.

The machine learning can be defined as a ability to extract patters from data without explicitly being programmed to. This is important difference to systems where computer or program has hard-coded knowledge of a patterns such as rules of a game or how a dog look like. This allowed to tackle various real life problems whit many different solutions which wouldn't be able to solve using hard-coding. Such system can be used in object recognition, natural language processing, email filtering or finance.

The machine learning as a field is a part of the computer science. Concretely, it is a sub-field of broader field of artificial intelligence (Goodfellow, Bengio, and Courville 2016). Machine learning tasks are usually divided into two[1][2] broad categories:

- Supervised learning – learning by a teacher requiring inputs and outputs
- Unsupervised learning – learning or finding new patterns in data given only inputs

[1] There is also a mix of the two called semi-supervised learning

[2] There is also task called Reinforced Learning

2.1.1 Supervised Learning

This machine learning task is very common. It requires N examples of input/output pairs $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_N, y_N)$, where x_j is some set of features and y_j is a target. We assume that y_j is generated by some unknown function $y = f(x)$ using x_j. We also assume that the function is hidden and will stay that way. The goal is to find this generating function. The solution used in supervised learning is to approximate this function by another function h.
The h function is a particular hypothesis about the generating function and it is a part of space of possible hypotheses $\mathcal{H}$. The process of search of $\mathcal{H}$ for the right hypothesis h can be defined as learning. The goal is to select particular hypothesis that will fit out training data well. Such function can be used later on examples pairs outside the training set. This means that the function will predict correct y using x even in cases that where not presented before. This ability is called **generalisation**. The target can generally come from two sets. A task where y is from finite set of possible values is called **classification**. It is called binary classification when the finite set has only two values. If y is a number, the task is called **regression**. Usually in regression, the target is some real number values. Common supervised learning algorithms are: Linear Regression, Support Vector Machines, Neural Networks, Decision Trees and many more. Supervised learning is used in this thesis. Concretely, we use neural networks algorithm for regression of option prices.

2.1.2 Unsupervised Learning

Unsupervised Learning is less complex than supervised learning. It is given by fact that unsupervised learning is usually used for tasks where there are no targets y and only x are available. Unsupervised algorithm can thus use only x and it is given no feedback about it's performance. Very common task for unsupervised learning is **clustering**. It means grouping objects into groups so that each object in a group is similar to other objects in the group. Another common task for unsupervised learning is **dimensionality reduction**. In this task, the algorithm tries to reduce size of given object while preserving relevant structure in it. Common unsupervised learning algorithms are: h-means for clustering, special kind of neural networks such as auto-encoders or generative adversarial networks and for dimensionality reduction Principal Component analysis.

2.2 Artificial Neural Networks

An *artificial neural network* (ANN) is a main tool used in this thesis. As the name suggests, ANNs were partly inspired by biological brain so they try to mimic learning of the biological neuron fig. 2.2. Human brain has approximately 86 billion of such neurons so it be cannot be replicated so far (Herculano-Houzel and Lent 2005). Each neuron is capable of receiving, processing and outputting information to another neuron. Neuron receives signals using it's dendrites and outputs signal using axons. Neurons are connected though synapses each has some strength, which allows to transfer information from axon to another neuron's dendrite.

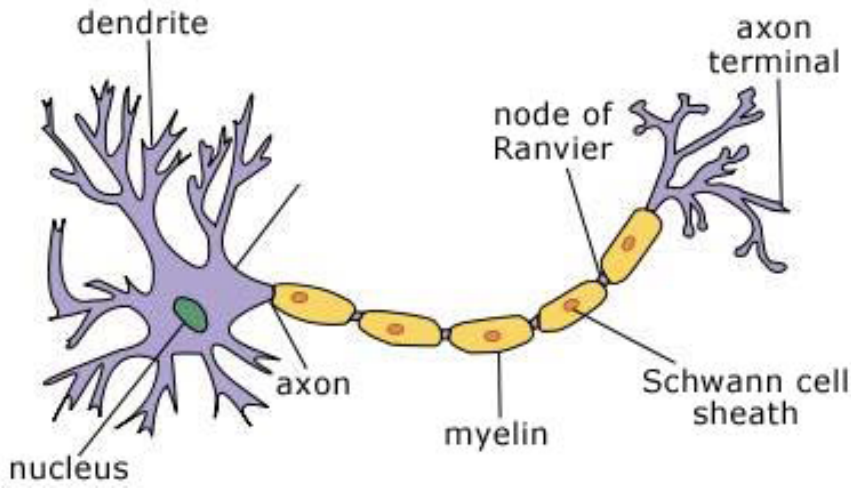


Figure 2.1: Biological neuron
Source: `http://www.e-missions.net/cybersurgeons/?/nerv_student/`

In general, ANN consist of **artificial neurons** (or nodes). ANN can be described as connections of these nodes (artificial neurons or units) that are simple versions of neurons in human brain. These artificial neurons are usually grouped in layers. Neurons in each layer are connected to neurons in other layer via simple version of synapses. These connections has weights and they allow to transfer information from one neuron to another and consequently from one layer to another. The information received may be then transformed by non-linear function and then transferred to another neuron. The most simple neural network consists of only one neuron fig. 2.2.

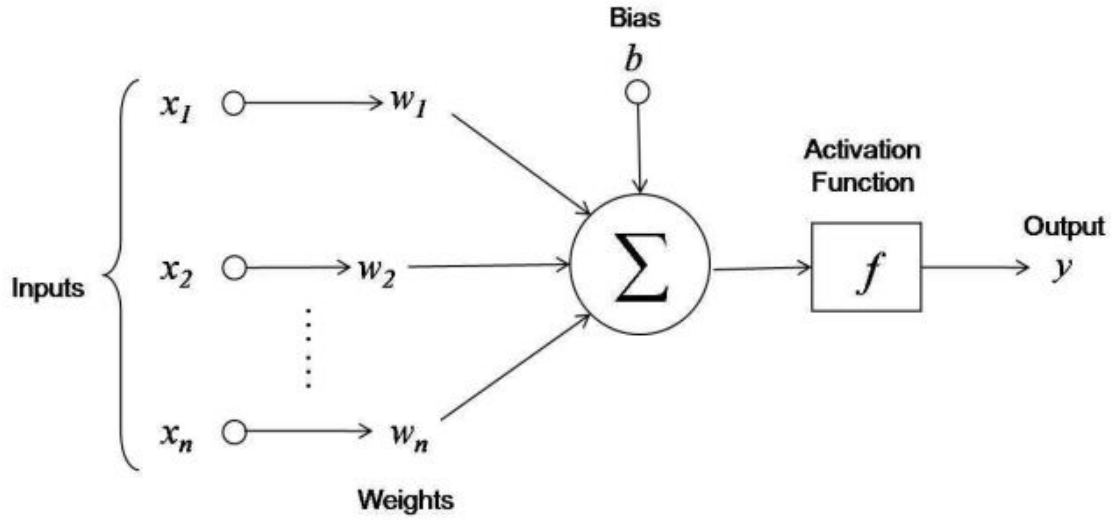


Figure 2.2: Simple network with one neuron
Source: `http://blogs.cornell.edu/info2040/2015/09/08/neural-networks-and-machine-learning/`

Each neuron has some inputs $x_1, x_2, \ldots, x_n$, that are combined using some weights $w_1, w_2, \ldots w_n$ and bias b. The potential of network fig. 2.2 is:

$$\xi = \sum_{i=1}^{n} w_i x_i + b$$

If we consider activation function f than output of such network is:

$$y = f(\xi) = f(\sum_{i=1}^{n} w_i x_i + b)$$

The output can serve as an input into another layer of neurons or as an output. In general, there are many neurons arranged into layers. There are usually three kind of layers: input layer, hidden layer and output layer. ANN with only one hidden layer is called single layer neural network or "vanilla" neural network.

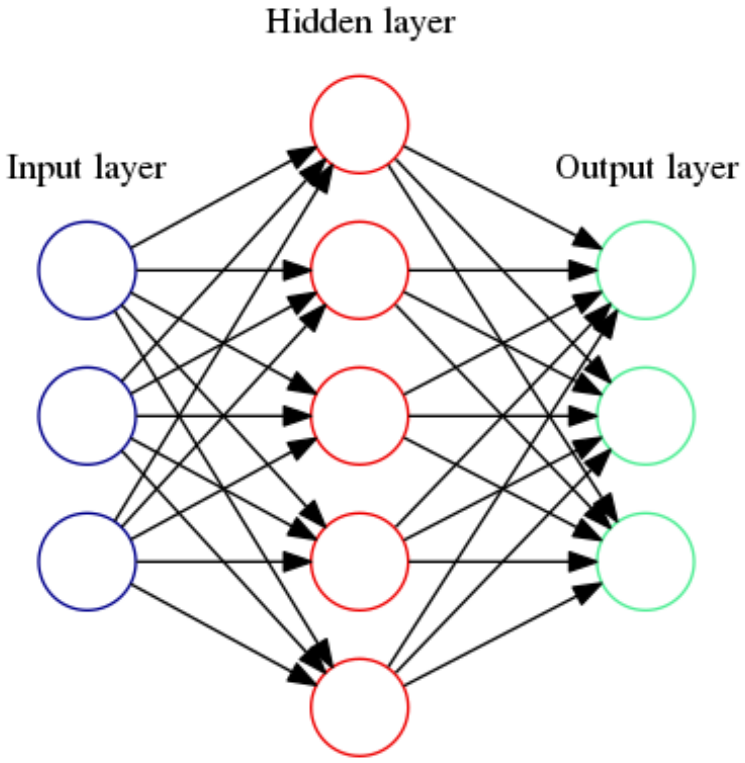


Figure 2.3: Neural network architecture diagram with one hidden layer

ANN is usually a two-stage regression or classification model, which can be organised into many architectures. Blue circles $X_p, p = 1, \ldots, P$, represent input neurons. Red circles $Z_m, m = 1, \ldots, M$, represent neurons in hidden layer and green circles at the top $Y_k, k = 1, \ldots, K$ represent output neurons. For regression, typically $K = 1$ so the network computes only one Y_1 unit at the top. Usually, it is a real number and the neuron is activated using the identity output (or activation) function. However the identity function doesn't not effectively transform the output so one can omit it. For classification, $K = 2$ or more so the network has K units at the top and where Y_k is computing $k - th$ class or label.

Features Z_m in the hidden layer are computed as linear combinations of the inputs and Y_k units in the output layer are linear combinations of Z_m as follows:

$$\begin{aligned} Z_m &= \sigma(\sigma_{0_m} + \alpha_m^T X), m = 1, \ldots, M, \\ T_k &= \beta_{0_k} + \beta_k^T Z), k = 1, \ldots, K, \\ f_k(X) &= g_k(T), k = 1, \ldots, K, \end{aligned} \tag{2.1}$$

where $X = (X_1, X_2, \ldots, X_P)$ represents vector of inputs, $Z = (Z_1, Z_2, \ldots, Z_M)$ rep-

resents vector of derived features, $T = (T_1, X_2, \ldots, T_K)$ represents vector of outputs before a final transformation by output function $g_k(T)$ and where σ is an activation function.

Neural networks are usually designed with addition neurons called **bias** σ_{0_m} and β_{0_k} in equation 2.1. They represent intercepts in linear combinations and they allow to shift the activation function to the side so the model can learn faster.

The output function $gk(T)$ depends on the task modelled. For regression, identity function $gk(T) = Tk$ is usually picked. For classification, the output function depends on number of classes. If $K > 2$ than Soft max function is usually picked and for $K = 2$ or binary classification the sigmoid function is picked. More in section2.4

One of the most important advantage of neural network is their extreme flexibility. As shown by Cybenko "any continuous function can be uniformly approximated by a continuous neural network having only one internal, hidden layer and with an arbitrary continuous sigmoidal non-linearity" (Cybenko 1989) for sigmoid activation function. And latter showed by Hornik (Hornik 1991) that standard multilayer feedforward neural network with only one hidden layer and arbitrary activation function is **the universal approximator**. Hornik however assumes that the activation function is bounded. This would prohibit the ReLU function see section 2.4 which is very popular and has properties. However recently it was showed by Murata & Sonoda (Sonoda and Murata 2017) that the approximation theorem holds for unbounded activation functions too. This means that it is possible to approximate function basically any function. It can be function that governs car riding on a street, function that recognise a human being on a picture or function that distinguish a dog from a cat. We may conclude that neural networks can approximate even function for pricing arbitrary option with arbitrary properties. It is however wise to acknowledge that the universal approximation theorem does't deal with feasibility. One has to use very high number of neurons and layers in order to proximate real life function. In fact if we would like to approximate arbitrary function infinitely close we would need to have infinite number of neurons in the hidden layer.

Neural networks can be compared to traditional parametric models. These models require that a nature of relation between independent variables and a dependent variable is specified e.g. linear or exponential. However if the neural network has at least one hidden layer than it is capable to develop an internal representation of the relationship between variables. This mean that there is no prior assumption about distribution of parameters required (White 1989).

Another advantage of ANNs is the fact that they have no assumption about

data itself. Only standardisation of input features is recommended, but it helps the network to learn faster.

2.3 Multi-layer perceptron

We use more more complex architecture of ANNs than single layer feed-forward neural network described in previous section. And that is **Multi-layer perceptron** (MLP) and consequently with Recurrent neural network (RNN). The MLP differs from single layer neural network in number of hidden layers. The MLP has more than one. Both are feed-forward, which means that the information flow from input layer to output layer in acyclic way so the information is never delivered back to previous layer. This property has it's advantages, but also disadvantages. The network is no able to preserve information e.g. it has no memory, but it's training is faster and easier. Lack of memory can be overcomed by allowing the network to send back some information from higher layers to lower layers. This can be done multiple times on multiple level. Such networks are called **recurrent neural networks** and they will be topic of another section. MLP are usually fully connected. It means that every neuron is connected to every neuron in following layer and there are no missing connections.

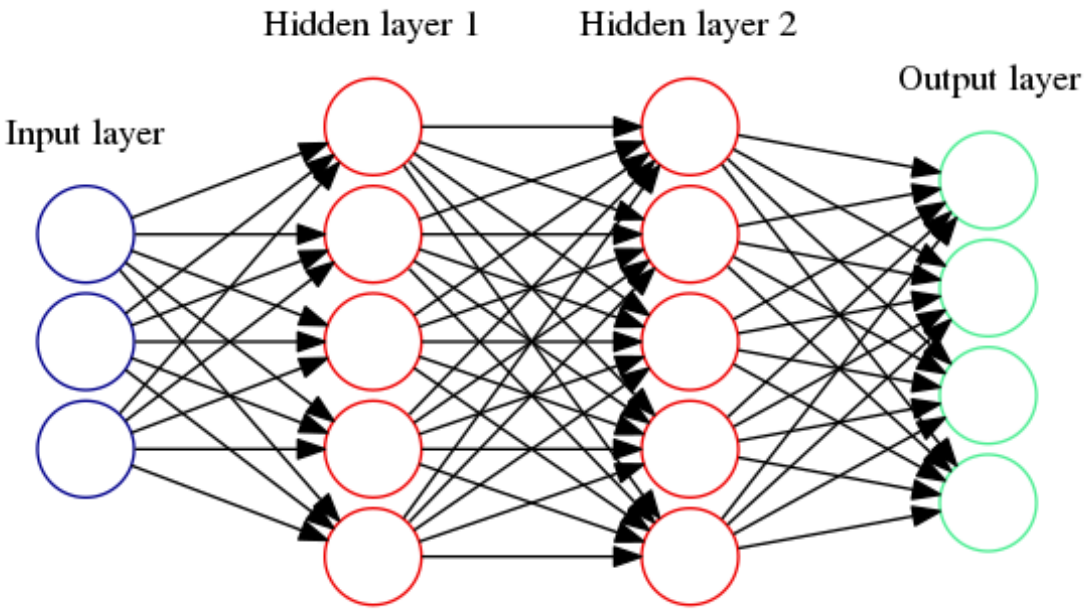


Figure 2.4: MLP with two hidden layers

2.4 Activation Functions

In the previous sections we mentioned **activation functions** as a functions that transform inputs into outputs in a neuron. They were also identified as a key source of non-linearity in any ANN. If the activation function σ is identity function then it would cause neural network to become linear in inputs. So the non-linearity is given to the model by non-linear activation function.

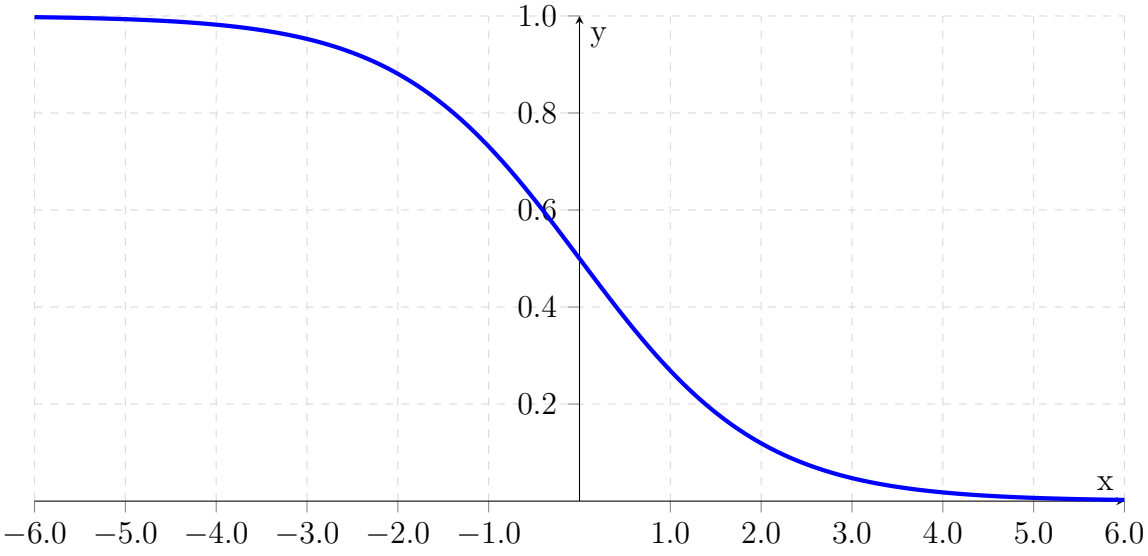
In general, there are many activation functions that differ in possible output values and other properties. We will discuss here a few key functions.

2.4.1 Sigmoid Function

In general, a **sigmoid** function is a monotonically increasing function that maps real values into values between usually 0 or 1 or -1 and 1.

One example is a standard logistic function or logit:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



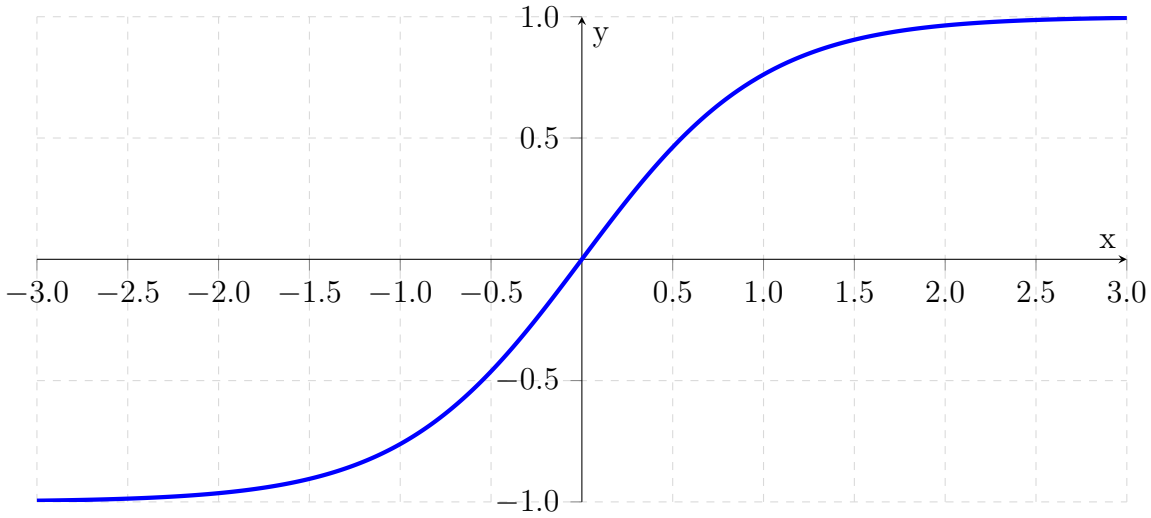
This function has been historically very popular. However recently this function has been replaced by different functions due to shortcomings of this function. If we assume that an input to the neuron is either very high or very lower then the gradient of this function in backpropagation[3] will be very small and effectively restraining the neuron from learning[4]. Nevertheless this function is still used in binary classification problems in the output layer or in different tasks.

[3] More in next section

[4] vanishing gradient problem

Another example of a sigmoid function is the **tanh** function:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

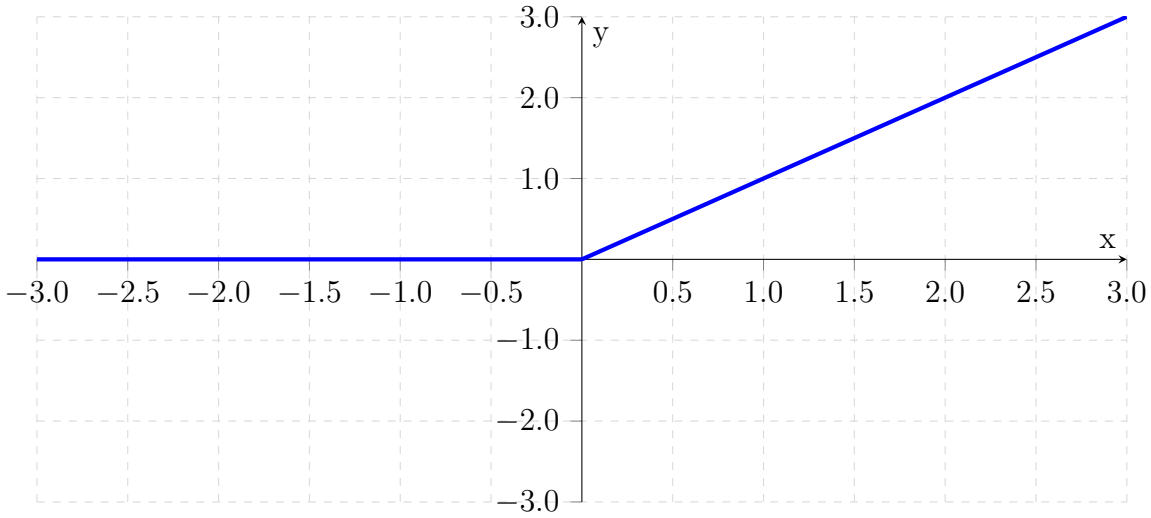


This function is basically scaled version of the logistic function. However the tanh is able to over come the diminishing gradient problem and it is able to converge faster than simgoid in general (Lecun et al. 2012).

2.4.2 Rectified Linear Units

Rectified Linear Units (ReLU) is an activation function that is defined by:

$$f(x) = max(0, x)$$

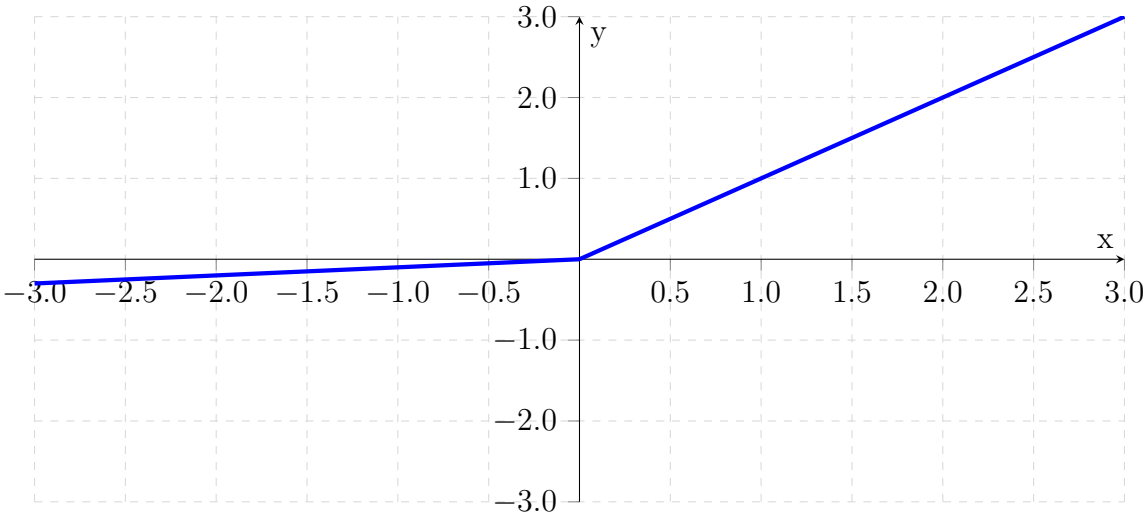


It is a non-linear function which remains very close to linear function and in a sense it consists of two linear functions. In spite of it's simple structured, it has been

found that the ReLU is able to accelerate a convergence of the network in comparison to the sigmoid functions (Krizhevsky, Sutskever, and E. Hinton 2012). Also it has no "expensive operations" such as (exponentials etc.) in sense of computing power or memory requirements. It is recommended as a default activation function for MLPs (Goodfellow, Bengio, and Courville 2016). However even this function is not without drawbacks. Sometimes this activation function causes neurons not to activate itself. So there has been an updated version called **Leaky ReLU** (Maas, Hannun, and Ng 2013) which has slightly modified formula:

$$f(x) = max(x, \alpha x)$$

where α is a constant, usually small one.



There are many more activation functions, but it is out of scope of this paper to introduce all of them.

2.5 Learning and Parameter Fitting

The ANN has unknown parameters called weights and biases mentioned in the previous section. The most important feature of the ANNs is their ability to learn these weights and biases so they fit a specific environment that is introduced to the network. This process is iterative. In every iteration output of network is compared to the environment observed and the weights and biases are updated accordingly. So each iteration should yield output more similar to the environment until desired point. Learning however might exceed the point and the network starts to learn an error specific for the environment instead of general knowledge. This is called **over-fitting**.

We are not looking for point where the output is the most similar to the environment during training, but to the point where the network has learn all the knowledge it could. In other words, we are not looking for global minima, but particular local minima. It is necessary to use some kind of regularisation technique, that will penalise the network for such behaviour and lead to the local minima. Before we continue with explanation of learning algorithm in details, it is worth to mention specifications for supervised learning. As already mentioned in section 2.1, ANNs belong to family of supervised learning algorithms. To use these algorithms, one need to split dataset used for modelling into sets. Basic division of the data is into **training set** and **test set**. The training set is used for learning or finding best weights and biases. The test set is used for calculation overall performance of the network. It is however feasible to split the test set into two smaller sets: **validation set** and test set. The new set, validation set, is used for selecting best model and it's hyper-parameters. In case of neural network, it is used for example for selecting number of hidden layers and neurons. More in Methodology chapter. Moreover the difference between the error in validation set and the error in train set is used for monitoring of learning process and subsequently for estimation of over-fitting. In general, if the difference is significant than it is probable that the network is over-fitting.

The error is usually calculated by the error function. There are many measures that can be used as an **error function**, but we will continue with sum of squared errors which is very common for regression:

$$R(\Theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{i_k} - f_k(x_i))^2 \tag{2.2}$$

where Θ is set of weights:

$$\begin{aligned} \{\alpha_{0_m}, \alpha_m; m = 1, 2, \ldots, M\} \quad & M(p+1)\ weights, \\ \{\beta_{0_k}, \beta_k; k = 1, 2, \ldots, K\} \quad & K(M+1)\ weights \end{aligned} \tag{2.3}$$

The method used for learning is called *backpropagation*. It uses a generic approach to minimise the error function by optimisation algorithm called **gradient descent** (sometimes steepest descent).

Gradient descent (GD) is an algorithm that minimises a function that is given by set of parameters. The algorithm starts with some initial values of the parameters. The parameters are iteratively updated so the position is updated towards minimum. The updated is achieved by taking negative [5] direction of the function gradient times

[5] If the algorithm uses positive direction that it is maximising the function

some step size α.

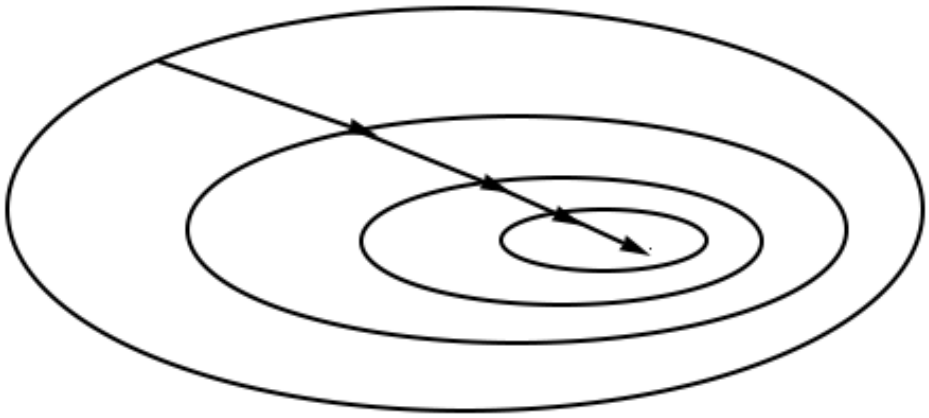


Figure 2.5: Gradient descent path

In backpropagation, the parameters are weights and the function being minimised is the error function. Given the structure of a neural network, the gradient can be found by applying **chain rule** for differentiation.

Let's take the error function $R(\Theta)$ from 2.2, initial set of weights Θ, $z_{m_i} = \sigma(\sigma_{0_m} + \alpha_m^T x_i)$ from 2.1 and let's $z_i = (z_{1_i}, z_{2_i}, \ldots, z_{M_i},)$ then:

$$R(\Theta) = \sum_{i=1}^{N} R_i = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{i_k} - f_k(x_i))^2 \tag{2.4}$$

and derivatives of 2.4 are:

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{k_m}} &= -2(y_{i_k} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{m_i}, \\ \frac{\partial R_i}{\partial \alpha_{m_l}} &= -\sum_{k=1}^{K} 2(y_{i_k} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{k_m} \sigma'(\alpha_m^T x_i) x_{i_l}. \end{aligned} \tag{2.5}$$

Now given derivatives 2.5 a gradient descent update of the weights in step (r+1) has the form as follows:

$$\begin{aligned} \beta_{k_m}^{(r+1)} &= \beta_{k_m}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta^{(r)_{k_m}}}, \\ \alpha_{m_l}^{(r+1)} &= \alpha_{m_l}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha^{(r)_{k_m}}}, \end{aligned} \tag{2.6}$$

where γ is *the learning rate* e.g. step size in a gradient descent.

If γ is high e.g. $1e^{-1}$ then training is going to be relatively fast, but there is a risk of over-shooting which might cause divergention of weights and thus poor performance of the network. If the learning rate is relatively small e.g. $1e^{-3}$ then parameters found are going to be more reliable, but the training may take a lot of time. This problem is usually overcommed by using different constants for different phases of training. At the beginning γ will be high e.g. $1e^{-1}$ and it will be gradually changed for smaller values along the training to e.g. $1e^{-5}$.
If 2.5 is written as:

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{k_m}} &= \delta_{k_i} z_{m_i}, \\ \frac{\partial R_i}{\partial \alpha_{m_l}} &= s_{m_i} x_{i_l}. \end{aligned} \tag{2.7}$$

We can refer to δ_{k_i} and s_{m_i} as errors at output and hidden layer, respectively. The errors must satisfy:

$$s_{m_i} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{k_m} \delta_{k_i} \tag{2.8}$$

The equation 2.8 is known as the backpropagation equation. It allows to update 2.6 in a two-pass algorithm. In the first pass, the current weights are frozen and then used in 2.2 for predicting $\hat{f}_k(x_i)$. This is called the **forward pass**. In the next pass, the output layer error δ_{k_i} is computed and using 2.8 back-propagated to give the hidden layer error s_{m_i}. Then δ_{k_i} and s_{m_i} are used for computation of updates in 2.6. This is called the backward pass. When new set of weights is computed, the whole process starts again so the weights are frozen and predictions are made... This process is know as backpropagation. Earlier, it was also called the **delta rule** (Widrow and Hoff 1960). The backpropagation is simple and very suitable for parallel computing architecture. For detailed explanation of backpropagation please see (Hastie et al. 2004).

2.5.1 Gradient Descent

Gradient descent algorithm used in backpropagation section is only basic or "vanilla" version (Karpathy 2018). In this version, weights are updated only using gradients and learning rate. However there are other versions of GD. One is **Mini-batch gradient descent**. In vanilla GD, the error loss is computed after the entire training set was processed and only after that the weights are updated. In mini-batch GD, the error is computed after batches of training data examples. The mini-batch is usually set of size in power of 2 e.g. $8, 16, 128, 256$ etc. This is because

of computer memory constraints allows faster vectorized computation in this way. The gradient in mini-batch algorithm can be considered as a good approximation of the gradient computed using full training data. Another version is called **Stochastic Gradient Descent** (SGD). Sometimes it may be referred as on-line learning. In SGD, the mini-batch has size 1 and it is randomly selected from training set without replacement. In practice, this is not very common to use only one example at a time, because such training is not very efficient. The best option is to use SGD with batch size higher than 1. Also to be consistent, GD used in previous section can be referred as batch GD, with batch size equal to training set.

2.6 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a category of neural networks where connections between neurons (units) form a direct cycle along some sequence. This allows it to display dynamic behaviour for the time sequence. There exists many different architectures of RNN such as Hopfield network (Hopfield 1982), which composes of system of binary threshold units, Elman network (Elman 1990), which has an additional set of units representing context or Long Short-term memory (LSTM) (Hochreiter and Schmidhuber 1997), which is going to be explained in more detail in following sections. RNNs have been successfully applied to many tasks such as handwriting recognition (Graves et al. 2009), speech recognition (Sak, Senior, and Beaufays 2014) or time series modelling as they are able to approximate time series very well (Anbazhagan and Kumarappan 2013).
The cycle allows the network to memorise information, in a **memory cell**. The network can keep relevant data in the cell though time and use this information in right time in the future. For example in text translation, the network can remember that the word network in this sentence is in singular and it may decide to remember this information so it can continue text prediction in singular. This makes RNNs in principle more skill-full than MLPs. MLPs are only able to map inputs to outputs, but the order in which the network is presented with examples is irrelevant. In fact, it is recommended to shuffle the training examples before the training of MLPs. RNNs however are capable to gain information from history of previous inputs thus the order of examples is crucial. It is fair to say however that if RNN would only have been introduced with sequence of length one than the performance would be comparable to MLP as with only one time-step there is no history so the RNN looses it's advantage. If one would like to use MLP for sequence then she would have to

input the history as lagged values in a sequenced form.

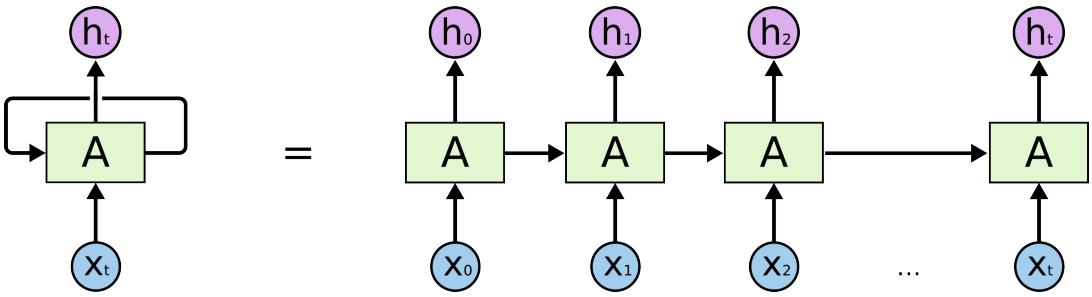


Figure 2.6: RNN

Source: `colah.github.io/posts/2015-08-Understanding-LSTMs`

The RNN is visualised on fig. 2.6. We can see on the left side RNN in unrolled state, where the loop arrow represents recurrent flow of information between units. On the right picture, we can see the same network in rolled state where each box represents an unit of the network. Both diagrams shows the same network. Only that one is unrolled and second in rolled state which has no impact in real word as it is only an abstraction.

RNNs similarly to MLP have ability to approximate any measurable function arbitrary well given sequence input to sequence output (Hammer 2001).

2.6.1 Vanishing Gradient Problem

The early version of RNNs were not very popular and their training was very long and difficult (Pascanu, Mikolov, and Bengio 2012) not only due to lack of nowadays computation power, but also due to the **vanishing gradient problem** (Bengio, Simard, and Frasconi 1994)[6]. The vanishing gradient problem is a undesired phenomenon that occurs in training of ANN with gradient-based methods such as backpropagation. The problem is aggravated when the network has many hidden layers. Gradient based methods learns their weights by receiving updated proportional of the gradient of the error function with regard to current weights. If a change in the weights causes very small change in output than the network is unable to learn rights weights in effective way. The problem occurs when the gradient is so small that it effectively prevents the weights from being updated. One of causes of the problem is choose of the activation function. If the network uses the sigmoid function than its gradients are in range $(0, 1)$ or $(-1, 1)$ so there are regions in input

[6]The authors also describes opposite problem called exploding gradient problem

space where relatively large change in input will cause small change and thus small gradient. The problem is aggravated when this small gradient is multiplied by n small numbers in m-hidden layers so the gradient decrease exponentially causing the front layer weights to train extremely slowly or not at all. The problem was overcommed for RNNs by introduction of **Long Short-term memory** (LSTM). For MLPs the problem is commonly overcommed by using special activation functions such as ReLU or Leaky ReLU, more in section 2.4.

2.7 Long Short-Term Memory

Long-short term memory (LSTM) units are building block of the RNN network called the *LSTM network* and were proposed by Hochreiter and Schidhuber (Hochreiter and Schmidhuber 1997) and improved few years latter (Gers, Schmidhuber, and Cummins 2000). The network is capable of holding information for a long time. It was designed to have ability to learn long term dependencies. This is done by a memory cell which causes that error going through remains constant over time. Every memory cell is capable of reading data, writing data and even erasing data from it. These memory cells or units are usually arranged into blocks, where units are fully interconnected which allow them to share information between them. The blocks can be arranged into layers. LSTM network can have multiple such layers.

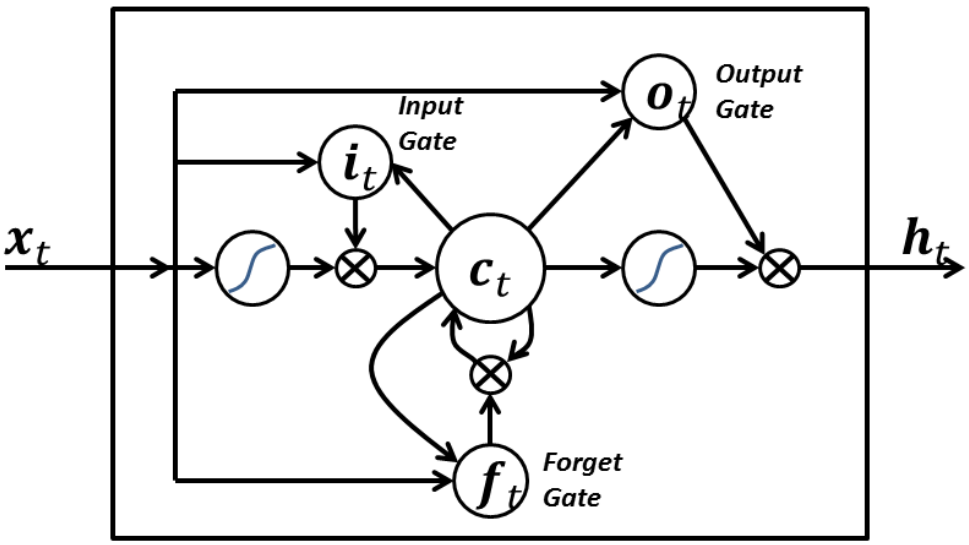


Figure 2.7: LSTM unit
Source: `github.com/OKStateACM/AI_Workshop/wiki/LSTM-Generator`

Each LSTM unit has usually four parts: a cell, an input gate, an output gate and a forget gate. The gates are structures governing the processes of reading, writing and erasing. The gate is very similar to a single neuron in feed-forward neural network. It computes weighted sum of inputs and then they activate the sum using activation function. They have a role of valves that regulate a flow of information that goes into and out of the cell. The original structure proposed by Hochreiter and Schidhuber had no **forget gate** (Gers, Schmidhuber, and Cummins 2000). The gates have usually the logic activation function which maps real values into $(0, 1)$. If the gate output has value close to zero than no information is let through. If the gate has value close to one than almost all information is let through.

2.7.1 Forward Propagation of a Single Unit

Forward propagation of a single unit is explained in more details using notion as in (Goodfellow, Bengio, and Courville 2016). The input of the single unit consist of network input $\vec{x}$ and $\vec{h}$ which is hidden layer vector. If the network has multiple layers than $\vec{x}$ can be output of previous layer. If the unit is very first in the block than $\vec{h}$ is randomly initialised. The target is to update the cell $s_i^{(t)}$ and to get output $h_i^{(t)}$.

The pass starts with a forget gate $f_i^{(t)}$ for time step t and cell i which is a value between 0 and 1:

$$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}), \tag{2.9}$$

where σ is a logic sigmoid activation function, $\vec{b}^f$ denotes biases, $\vec{U}^f$ denotes input weights and $\vec{W}^f$ denotes recurrent weights for the forget gate. Afterwards, the cell state $s_i^(t)$ is updated using $f_i^{(t)}$ and $s_i^{(t-1)}$:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}), \tag{2.10}$$

where σ is a logic sigmoid activation function, $\vec{b}$ denotes biases, $\vec{U}$ denotes input weights, $g_i^{(t)}$ is an input gate and $\vec{W}$ denotes recurrent weights for the inputs. The input gate $g_i^{(t)}$ is computed using it's own parameters:

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}) \tag{2.11}$$

where σ is the logic sigmoid activation, $\vec{b}^g$ denotes biases, $\vec{U}^g$ denotes input weights and $\vec{W}^g$ denotes recurrent weights for the input gate. At the end, the output $h_i^{(t)}$

computed using the output gate $q_i^{(t)}$:

$$h_i^{(t)} = tanh(s_i^{(t)}) q_i^{(t)}, \tag{2.12}$$

where $tanh$ is the tahn sigmoid activation function and $q_i^{(t)}$ is computed using output parameters:

$$q_i^{(t)} = \sigma(b_i^{\circ} + \sum_j U_{i,j}^{\circ} x_j^{(t)} + \sum_j W_{i,j}^{\circ} h_j^{(t-1)}) \tag{2.13}$$

where σ is a logic sigmoid activation, $\vec{b}^{\circ}$ denotes biases, $\vec{U}^{\circ}$ denotes input weights and $\vec{W}^{\circ}$ denotes recurrent weights for the output gate.

The above equations describe how a single unit is updated in one forward pass and how the output values are computed. In practice, there would be several units in a blocks which would share information, but not weights. The weights are trained using recurrent version of backpropagation algorithm.

Chapter 3

Methodology

Different methods we used for price prediction of the options are summarised in this chapter along with other technical details of implementation and estimation. We have decided to use and compare three different methods for pricing options. We use **same inputs** to all the methods bearing in mind our main goal. If we would use different inputs for different methods we won't be able to prove that LSTM networks are suitable for option pricing when there would be information asymmetry between compared methods. Also we want to keep inputs simple and easy to access in real world situations. We use data described in data chapter as inputs. Proposed approaches are:

- As the benchmark, we are using Black-Scholes model. We use pricing formulas proved in previous chapters for calculating prices.

- The next approach is feed-forward neural network. Concretely, we use MLP with same inputs as Black-Scholes formulas one output representing price of the option.

- The finale approach is to use recurrent neural networks. Concretely, we have used LSTM network with the same inputs as the Black-Scholes and MLP and with one output unit representing price of the option.

Additionally, we have decided to use two approaches for estimating volatility. The first one we call historical volatility and the second **realized volatility**. Thus in total we compare six models where three are using historical volatility and other three are using realized volatility. Process of volatility estimation is explained volatility section in data chapter.

3.1 The LSTM Netowks in Finance Modelling

There have been some research done on RNNs in financial time series forecasting, for instance (V. Kondratenko and Kuprin 2003) and (Bao, Yue, and Rao 2017). A lot of authors have used recurrent networks and LSTM networks for stock prediction or for prediction of different time series such electricity consumption or solar power generation. However we have not found any paper that would use LSTM networks for option pricing. So we propose following approach.

Given that we want to prevent the information asymmetry we only use inputs same as for BS and MLP. Also we want to take an advantage of time series nature of an option price. When one uses MLP network for option pricing the time series nature of the option price is not utilised, because the input is usually only snap of latest information in some point in time, but history up to this point is not presented. It is possible to input also history into the MLP, but this would mean to use varying and relatively big number of inputs.

We assume that it would be more natural to predict tomorrows price given all the information until now in time respecting order. Thus we use LSTM network as it is able to use time series and make prediction given some time series. We use the same 5 inputs as in MLP and the Black-Scholes formulas, however we transform every daily close price of single option (one sample) into shape that would allow us to use historical data. The transformed sample looks like following matrix:

$$\begin{Vmatrix} \sigma_1 & K & r_1 & T_1 & S_1 \\ \sigma_2 & K & r_2 & T_2 & S_2 \\ \sigma_3 & K & r_3 & T_3 & S_3 \\ \sigma_4 & K & r_4 & T_4 & S_4 \\ \sigma_5 & K & r_5 & T_5 & S_5 \end{Vmatrix}$$

where last row represents last daily information and the first row represents information from very first trading day of the option. Using the matrix, we forecast close price for $t = 6$. The matrix grows by one row every day the option is traded. In training, validating and testing we use matrices with various number of rows. For particular option we would treat every matrix as unique sample so if the option is traded for twenty business days than we would have twenty different matrices and twenty different samples. Or more precisely we would have matrix that in $t = 1$ would have one row, in $t = 2$ it would have two rows with the first one the same as in $t = 1$ and one new row, in $t = 3$ the matrix would have 3 rows with the first two rows the same as in $t = 2$ and very first one the same as in $t = 1$ and one

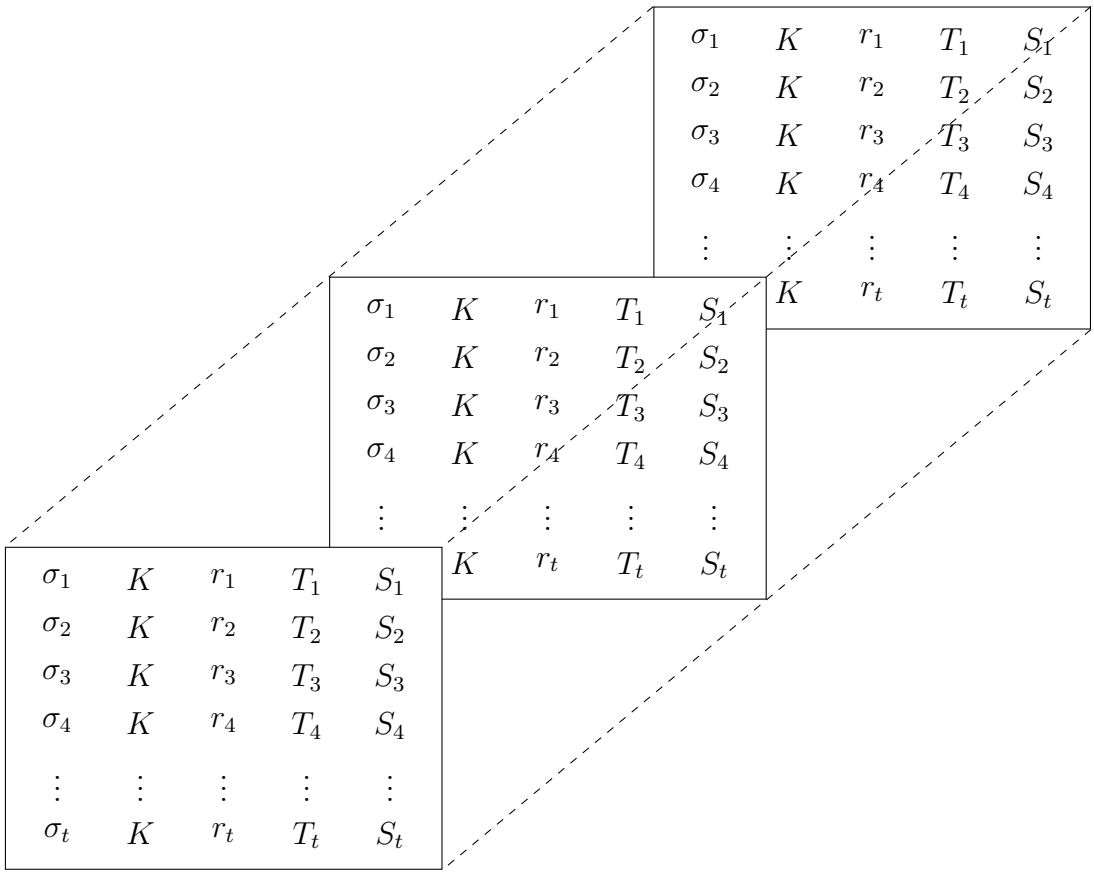


Figure 3.1: Tensor

new row and so on. It is obvious that K is going to be the same in every matrix given particular option as the strike price doesn't usually change during lifetime of an option. All the other inputs are going to change between different t.

We have transformed every sample in our dataset into the above matrix form. After this we concatenate all these matrices into 3-D matrix or *tensor*. The tensor have 3 dimensions representing number of inputs p, time-steps t and number of samples n. The tensor what than was than split into training, validation and testing tensor.

3.2 Hyperparameter Search

The most limiting factor of neural network and different suprevised algorithms is that they are parametrised by **hyperparameters**. The hyperparameters are used for configuration of a neural network and have wide varying effect on the resulting model performance (Claesen and Moor 2015). The hyperparameters are for exam-

ple number of hidden layers, number of neurons, learning rate, epoch, batch size etc. The problem is however that they are however not optimised by the network or algorithm itself. The process of looking for optimal hyperparameters is called **hyperparameter tuning** or **hyperparameter optimalization**. Same kind of model requires different optimal parameters given different dataset. Optimal hyperparameters are crucial in solving supervised learning task. Optimal hyperparameters yield highest/lowest score for given optimising metric.

The are several approaches for selecting hyperparameters. The most simple one is **grid search**. The grid search is hyperparameter optimalization approach that can be described as simple exhaustive search trough space of possible hyperparameters. Set of possible hyperparameters is usually set manually given some domain knowledge about the task being solved or dataset being used. Possible values for hyperparameters are organised into grid like structure where on x axis are possible values of one hyperparameters and and on y axis are possible values of different hyperparameters given only two hyperparameters. Finally, the algorithm iteratively tries every combination in the grid and outputs the set of parameters that achieved the highest/lowest score on the validation set. If there are more hyperparameters that the grid would have more axis. Another approach is **random search**. Random search compare to grid search doesn't exhaustively search every set of hyperparameters, but it take random values from hyperparameter space and tries these random sets. The space can be given by some discrete manually defined space as in the grid search or it can be continues or mixed space of values. When there are not many hyperparameters to optmize, random search is better option than grid search (Bergstra and Bengio 2012). Another option is to use **Bayesian optimization** which is a method used for optiomization of black-box functions. The method looks for optimal parameters by evaluation selected metric on validation set and map various sets of parameters into metric scores. The method similarly to grid search or random search, iterative tries combination of hyperparameters, but it not only measures their performance but it tries to gather information about the function being optimized and about location of it's minima. The key aspect of the method is that it used gathered information about the function to decide which set of parameters to try next. It was showed (Bernal, Fok, and Pidaparthi 2012) and (Hutter, Hoos, and Leyton-Brown 2011) that Bayesian optimisation needs less iterations to find generally better set of parameters than the grid search or the random search. We have used all three methods described above. We used combination of all three approaches and the best results where obtained by Bayesian optimization with grid searching specific are it has se-

lected. One disadvantage of our proposed approach is that we need to use additional parameter and that is the **timesteps**. We propose to use maximum length of the sequence. We believe that if we give the network as much historical information as possible than it can generalise better. It would be hovered feasible to experiment with different lengths. It might be the case that maybe only last 5 or 21 days are needed to price the options and additional information will only yield higher noise.

3.3 Technical Aspects

- **Batch size**: Represents number of samples that are propagated through the network. There is no optimal number that would fit all situations. Keskar et. al. (Nitish Shirish Keskar and Tang 2016) states that the optimal number of batch size is in rage between 32 and 512 samples. If the batch size is higher than there is a risk of degradation of model performance. If the number is smaller that computational time grows significantly. We have set the batch size to 128 for LSTM networks, because low number makes computation time very long. We tried to set bath size 1, but it would take 112 hours to complete one epoch. For MLP networks we used batch size = 8 given that MLP trains relatively fast.

- **Epochs**: One epoch represents propagation of all samples in the dataset. This parameter specifies how many times the algorithm will go through all samples in the dataset. Usually the more epoch the more accurate results, but more computational time needed. Given our computation time disposition we have decided to keep number of epoch low e.g. between 8 to 20. Ideally we would us higher number of epochs.

- **Hidden layers**: Number of hidden layers represents how many hidden layers there are in networks architecture. Usually more layers are used when one tries to solve complex problem such as image classification. The more layers mean that the network can capture more complex relationships between different features. LSTM network can also have more than one hidden layer. We have tried one and two hidden layers for architectures and we conclude that two layers are better in our case. We didn't try more layers as we believe that variables we put into the network are not that much complex that would require many layers and more layer would require more computation time. It is important to

keep in mind that more layers not always means better results, because more layers increase risk of over-fitting.

- **Number of neurons**: This parameter represents how many neurons or LSTM units there are in each layer. We have tried several options ranging from 5 to 300 Best combinations using historical volatility for MLP are 64 in first and 64 in the second layer for call options and 16 and 128 for put options. Best combination for LSTM networks was 16 and 64 for call options and 96 and 32 for put options. Best combination when using realized volatility are for MLP 128 and 32 for call options and 64 and 128 for put options. For LSTM network we used 96 and 32 for call options and 8 and 64 for put options.

- **Activation function**: This parameter represents activation functions used in network's architecture. We have decided to use default activation functions in LSTM network and ReLU in MLP. We have tried to experiment with output value in LSTM network, because it outputs negative values when the option price is small. However changing to ReLU or LeakyReLU prevent the network from converging. We recommend thus to add another layer after the output layer with LeakyReLU function with positive constant α. This would change negative values to small positive values. Such modification has yield some minor drop in RMSE and MAE. More about activation function in section 2.4

- **Error function**: Represent what kind of loss function the network use for optimization. There are many different loss function such as MSE, cross-entropy, hinge etc. Problem is that there is no generally preferred function. The choice as many other things in neural network depends on particular problem it's is solving. We have choose the MAE, given that our problem is regression and it's one of our measures.

- **Optimisation algorithm**: This parameter represents the algorithm used from optimization of the loss function. There are many choices. We have decided to use **ADAM** (Kingma and Ba 2014). We have experimented with many popular choices such as RMSprop or Adagrad, but without significant drop in any measure.

- **Dropout**: This is a regularization technique for reducing over-fitting. It is used in neural networks. It basically means to randomly drop some units during training neural network. We have tried various fraction of dropped units, but with no improvement in accuracy.

- **Input transformation**: It is feasible to transform input so they are all on the same scale. We use MIN/MAX feature scaling:

$$x_{scaled} = \frac{x - min(x)}{max(x) - min(x)}$$

where x is a feature.

3.4 Modules and Software

This section summarises data processing and software used in the thesis. Raw option data where downloaded from official web page of Montreal Exchange[1] in batches. Each batch has data about one calendar month. Together data consist of 63 batches. These batches were opened and parsed together into one big dataset table using Python. After that other data sources where parsed to the this table.

We have used following modules and software:

- **Python**[2] is a general-purpose programming language released in 1991. It has gained popularity in statistical and general data analytic in recent years. Together with R[3] it is the most popular language for data analytic. It has very powerful mathematical libraries such as NumPy and SciPy. Moreover, it has become popular language for neural network modelling. There are many popular libraries for training neural networks in Python such as PyTorch, Keras, Theano, TensorFlow, Caffe etc. Many of these are provide by companies like Google, Facebook or Microsoft which continuously develop these libraries. We have used Python version 3.6. We select Python because of all the above mentioned and because of it's flexibility and because it is open source.

- **NumPy**[4] is a powerful open source library for scientific computing in Python. One of it's main advantage are it's N-dimensional arrays. These arrays provide easy and very flexible way for operating with large matrices, tensors and other arrays. It also provides very nice variety of broadcasting functions. It's very special role in Python world can be proved by fact that many other frameworks maintain compatibility wit it. NumPy was used for storing our dataset and for generating inputs and outputs from and into networks.

[1] https://www.m-x.ca/nego$_f in_j our_e n.php$

[2] https://www.python.org/

[3] statistical programming language

[4] http://www.numpy.org/

- **Pandas**[5] is a library for data manipulation and analysis written in Python. It's DataFrame object is very suitable for working with structured data such as time series. It allows very convenient way for manipulation, filtering and sorting structured data. It allows to conveniently treat missing values and provide comfortable time series functions. These were used in computation of moving window variance in historical volatility. Pandas was also used for reading, parsing and writing the raw data batches. Moreover, it has additional features including summarising and plotting. Pandas is also open source.

- **scikit-learn**[6] is a library for machine learning written in Python. It has many ready to go algorithm for various task such as classification, regression, clustering etc. It also has very nice preprocessing functions. Some of them such as scaler for min-max scaling was used for reprocessing. Scikit-learn is designed to interoperate with NumPy and others. Also it is open source.

- **Keras**[7] is just like scikit-learn a machine learning library written in Pyhton, however Keras is more oriented to neural networks. Compare to other neural network libraries, it is designed to enable simple and fast prototyping of neural networks. It's key property is modularity. This means that a model is understood as a sequence or a graph of modules. These modelus could be put together with very few restrictions. It has also very broad range of function and utilities that help to use neural networks easily. We created our neural networks in Keras. Especially, LSTM units are easy to create and ensemble. However it also has some minor disadvantages. When Keras uses TensorFlow as backend, than is uses it's graph computation approach which requires that input into LSTM layer has to be in same shape in every sample. However in our dataset samples have different timesteps thus different length. This turned out to be a significant issue. Luckily, we have solved this by zero padding missing values in each sample so that every sample has the same number of timesteps. Only one sample has no zeros and it is the one that is the longest so all the other had to be padded to it's length. Also it is worth to mention that Keras's function *fit_generator* has allowed to generate samples in batches with no need of preloading to memory. This in turn has significantly eased memory demands for our computer. Keras is open source and very easy to learn and

[5]https://pandas.pydata.org/
[6]http://scikit-learn.org/stable/
[7]https://keras.io/

use. For example, simple neural network can be written in five lines of code.

- **TensorFlow**[8] is a library designed to high performance numerical computations. It is used for a range of task, but neural networks is one of the most important. Computation is based on the concept of tensors or multidimensial arrays. It uses dataflow graphs to represent different dependencies in individual computational operations. It means that user have to first define the dataflow graph and then create TensorFlow session to run this graph. We use TensorFlow as the backend for Keras.

- **Hyperas**[9] is a very simple and convenient wrapper for **Hyperopt**. Hyperopt is a python library that provides algorithms for Bayesian optimization (or sequential model-based optimization) for hyperparameter optimization of machine learning algorithms (Bergstra et al. 2015). Hyperas allows to use Hyperopt to Keras models. To use Hyperas one only needs to define the objective function to minimze and space over witch it will search optimal hyperparameters.

3.5 Forecast Measures

It is important to use right measures for evaluation in order to get robust results. We have picked two common used criterion's such as mean absolute error and root mean square error. Both are negatively oriented metric which can range from 0 to ∞. Both are based on compering model errors using the predicted values and the actual values. We do not use mean absolute percentage error (MAPE) as it is not suitable for values close to zero (Hyndman and Koehler 2006). And there are some option that have price between between 0 and 1 thus it wouldn't be optimal to use MAPE.

3.5.1 Mean Absolute Error

The Mean absolute error (MAE):

$$MAE = \frac{\sum_{t=1}^{n} |y_t - \hat{y}_t|}{n}$$

[8]https://www.tensorflow.org/

[9]http://maxpumperla.github.io/hyperas/

where y_t is an actual value at time t, $\hat{y}_i$ is a predicted value for time t and n is a size of the sample or the number of timesteps. The MAE has advantages of simple interpretation such as "the average magnitude of the errors", moreover it is robust and insensitive to outliers. Also it is suitable for forecasts on same scale(Hyndman and Koehler 2006), (Hyndman 2014).

3.5.2 Root Mean square error

The Root mean square error (RMSE):

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(y_t - \hat{y}_t)}{n}}$$

here y_t is an actual value at time t, $\hat{y}_i$ is a predicted value for time t and n is size of the sample or number of time steps. The RMSE is more sensitive to outliers than MAE. It has been popular historically due to it's theoretical relevance in statistical modelling (Hyndman and Koehler 2006).

Chapter 4

Data Description

4.1 Options

In this thesis we test out hypothesis on the S&P/TSX 60 Index Options (SXO) which are traded on Montréal Exchnage (MX). The MX is Canada's oldest exchange. It is fully electronic and dedicated to the Canadian derivative markets (Montréal Inc. 2018a). The SXO are European style and cash settled. According to exchange rules, minimum number of strikes is 5 bracking the spot price of an underlying asset(Montréal Inc. 2018b). This allow to get relatively high number of samples which is good for data hungry machine learning algorithms such as neural networks. Our dataset consists of options from 01.01.2013 till 30.03.2018, so we have more than 5 years of historical data. The data are daily close prices. There are 831173 options samples of call and puts together as per table 4.1.

Table 4.1: Options breakdown

Year	calls	puts	all
2013	55737	55737	111474
2014	73393	73393	146786
2015	76100	76113	152213
2016	97690	97690	195380
2017	90412	90412	180824
2018	22248	22248	44496
Total	415580	415593	831173

4.2 Stock price

The SXO options use the S&P/TSX 60 Index (SP/TSX60) as the underlying asset. The SP/TSX 60 is an stock index that represents leading companies in leading Canadian industries. It consists of 60 stocks of large capitalisation companies in Canada. Data were obtained from Thomson Reuters Eikon under ticket: *.SPTSE*. There are 1316 daily close prices of the SP/TSX60. Figure 4.1 shows the SP/TSX60 returns calculated by:

$$u_t = \ln \frac{S_t}{S_{t-1}},$$

where u_i is return in time i, S_i is a spot price in time t and S_{t-1} is a spot price in time $t-1$.

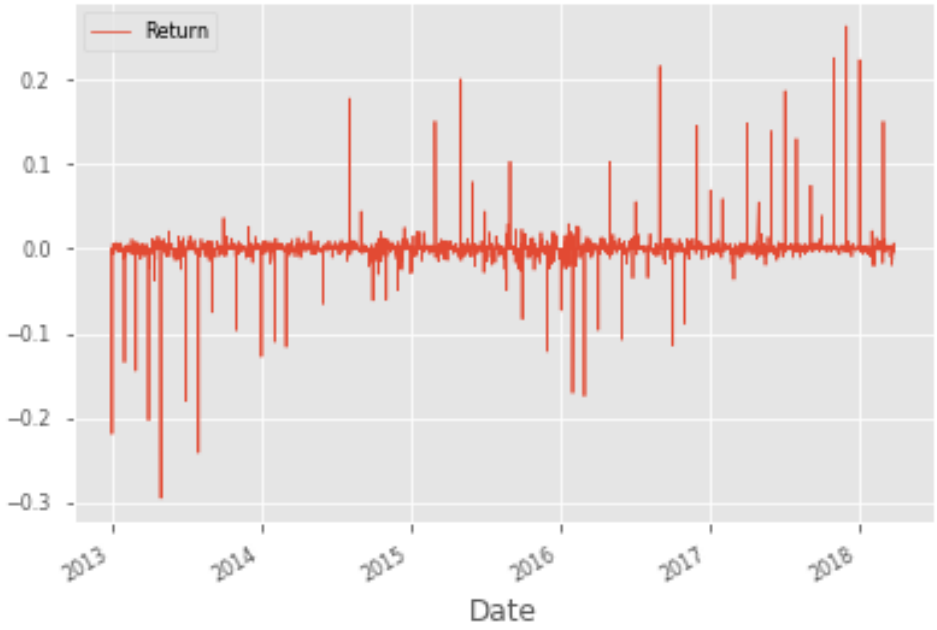


Figure 4.1: Returns

4.3 Interest Rate

We have picked Canada 3 Month Treasury Bill Yield as a proxy for risk-free rate. The data where obtained from Bank of Canada. It consists of 1310 daily close prices showed in fig. 4.2. We have choose 3 month term, because the average maturity of the options is 77 days.

Figure 4.2: Interest Rate

4.4 Volatility

4.4.1 Historical Volatility

Historical Volatility was estimated as a standard deviation of daily sample returns using returns computed in section 4.2 and 21 day long sliding window. We have followed process in (Witzany 2013) as follows:

$$s = \sqrt{\frac{1}{n-1}} \sum_{i=1}^{n} (u_i - \overline{u})^2,$$

where $n = 21$ in our case and

$$\overline{u} = \frac{1}{n} \sum_{i=1}^{n} u_i$$

Then we got annualised volatility as follows:

$$\hat{\sigma} = \frac{s}{\sqrt{\triangle t}},$$

where $\triangle t$ was set to $\frac{1}{252}$.

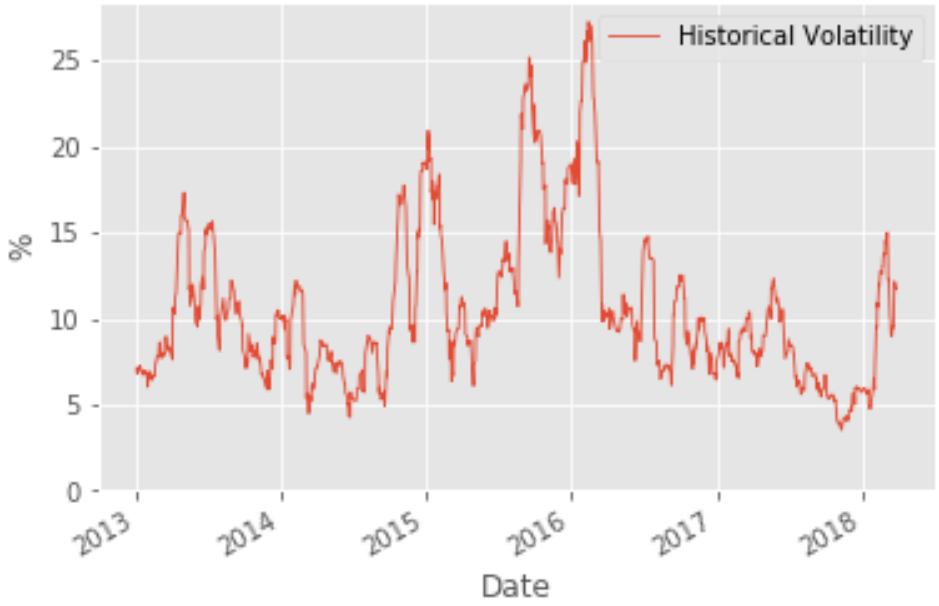


Figure 4.3: Historical volatility

4.4.2 Realized Volatility

Realized volatility is non-parametric estimator of volatility that takes advantage of information in intra-day (high-frequency) data (Andersen and Bollerslev 1998). The realized volatility can be defined as an aggregation of sum of squared returns over some small (down to 5 minutes) intervals for estimation usually daily volatility.

$$RV = \sum_{i=1}^{n} r_{i,n}^2,$$

where r is defined as logarithmic return in period from i to $i+1$. It would be theoretically possible to use i as small as possible in order to get almost perfect estimation of integrated volatility. This is however not possible to do so, because of effect of microstructure properties. We use 5 minutes period as these works better than 10 minutes period (Y. Liu, J. Patton, and Sheppard 2012). Data where kindly provided by Oxford-Man Institute of Quantitative Finance[1]. The raw data is in form of realized daily variance. The variance need to be squared and annualised in order to get daily annualized realized volatility. Also the Institute doesn't provide realized volatility directly for S&P/TSX 60, but for S&P/TSX Composite index which is closely related so we assume it as a good approximator.

[1] https://realized.oxford-man.ox.ac.uk/

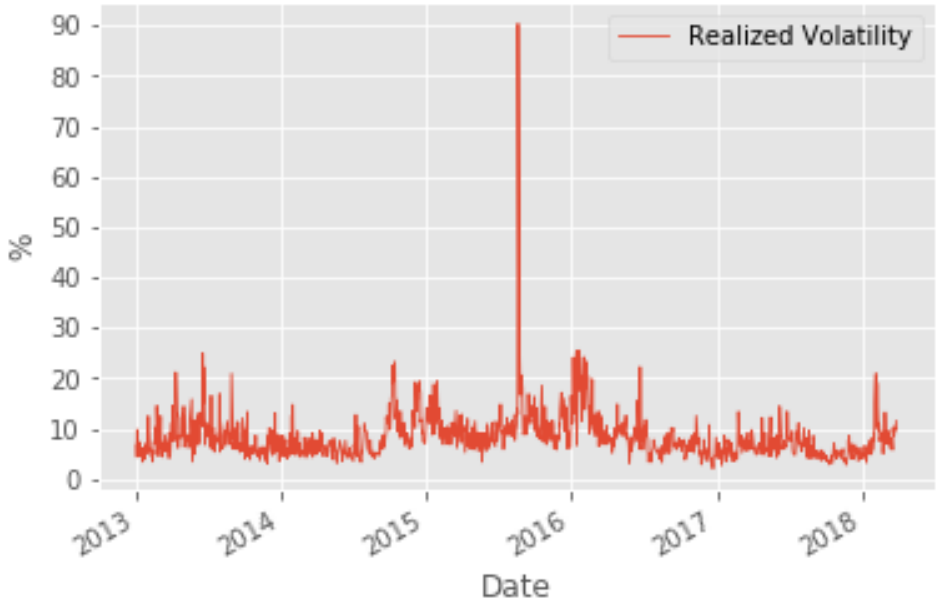


Figure 4.4: Realized volatility

4.5 Filtering and Division

We have applied several exclusion filters on out dataset in order to prevent some biases in option prices. Concretely, we have followed (Bakshi Gurdip 1997). First, we have excluded option with maturity less than 6 days. Then we have excluded option with quote price lower than 0.4CAD. Third, we have used condition from section 1.4 and excluded option that do not satisfy them. Forth, we have excluded options where one of features was missing or value was omitted due to other reason such as a bank holiday. After all this filtering, we have excluded more than 30% of original dataset.

These option where used for training, validating and testing models. Additionally we have decided to divide testing set into sub-samples regrading their moneyness and maturity length. This would allow more precise and comprehensive analysis. We again followed Bakshi, Cao, Chen (Bakshi Gurdip 1997). The option is considered to be ITM if the ratio between the spot price of the underlying asset to strike price $\frac{S}{K} > 1.03$, ATM if the ratio $\frac{S}{K} >= 0.97$ and $\frac{S}{K} < 1.03$ and OTM if the ratio $\frac{S}{K} < 0.97$. Regarding maturity, we define **short-term** options which expire in less than 60 days, **mid-term** options, which expire in more than 60 and less than 180 days and **long-term** options which expire in more than 180 days.

Table 4.2: Options breakdown after filtering

Year	calls	puts	all
2013	27933	43264	71197
2014	39635	57951	97586
2015	49461	59996	109457
2016	59675	79372	139047
2017	55357	72523	127880
2018	13161	17410	30571
Total	245222	330516	575738

Table 4.3: Test set division for calls

	short-term	mid-term	long-term
ITM	807	833	1198
ATM	1442	1143	750
OTM	4135	973	990

Table 4.4: Test set division for puts

	short-term	mid-term	long-term
ITM	4375	939	1611
ATM	1491	1134	720
OTM	4478	920	859

Chapter 5

Results

Below evaluations have been done using test set. The test set includes options from 04.01.2018 to 28.03.2018. The test set length represents roughly 5% of dataset. We present results for best hyperparameters found. We refer to Black-Scholes model as BS, Multilayer-perceptron as MLP and LSTM network as LSTM. We always predict next value in the sequence.

5.1 Historical volatility

5.1.1 Black-Scholes Model

Average RMSE for call options using BS is 160.849.

Table 5.1: RMSE for calls BS

	short-term	mid-term	long-term
ITM	47.24	85.2	315.25
ATM	4.18	45.97	393.72
OTM	31.63	113.42	235.7

Average MAE for call options using BS is 48.975.

Table 5.2: MAE for calls BS

	short-term	mid-term	long-term
ITM	16.4	38.02	156.52
ATM	1.99	27.51	221.13
OTM	6.29	40.73	103.54

The tables 5.1 and 5.2 summarise RMSE and MAE for call options prices using Black-Scholes formulas. The lowest RMSE is for short-term ATM options 4.18 and the highest for long-term ATM options 393.72. The lowest MAE is for short-term ATM options 1.99 and the highest for long-term ATM options 221.13. When we consider moneyness than ATM options have the lowest RMSE and MAE for short-term and mid-term options. OTM options have the lowest RMSE and MAE for long-term.

Average RMSE for put options using BS is 149.264.

Table 5.3: RMSE for puts BS

	short-term	mid-term	long-term
ITM	4.35	60.83	313.52
ATM	13.25	92.9	433.8
OTM	23.6	77.88	247.67

Average MAE for put options using BS 49.389.

Table 5.4: MAE for puts BS

	short-term	mid-term	long-term
ITM	3.64	31.83	173.85
ATM	9.62	55.88	258.66
OTM	13.2	50.81	140.38

The tables 5.3 and 5.4 summarise RMSE and MAE for put options prices using Black-Scholes formulas. The lowest RMSE is for short-term ITM options 4.35 and

the highest for long-term ATM options 433.8. The lowest MAE is for short-term ITM options 3.64 and the highest for long-term ATM option 258.66. When we consider moneyness than the lowest RMSE and MAE in short-term is for ITM options. For mid-term and long-term it is ATM options.

5.1.2 Multi-layer Perceptron

Average RMSE for call options using MLP is 127.641.

Table 5.5: RMSE for calls MLP

	short-term	mid-term	long-term
ITM	48	76.33	212
ATM	28.12	49.37	284.13
OTM	49.39	114.2	229.16

Average MAE for call options using MLP is 61.279.

Table 5.6: MAE for calls MLP

	short-term	mid-term	long-term
ITM	34.26	33.27	143.06
ATM	25.73	37.24	149.19
OTM	38.87	53.42	121.93

The tables 5.5 and 5.6 summarise RMSE and MAE for call options prices using Multi-layer perceptron. The lowest RMSE is for short-term ATM options 28.12 and the highest for long-term ATM options 284.13. The lowest MAE is for short-term ATM options 25.73 and the highest for long-term ATM options 149.19. When we consider moneyness than ATM options have the lowest RMSE for short-term and mid-term options and MAE for short-term. ITM options have the lowest RMSE for long-term and MAE for mid-term. OTM options have the lowest MAE for long-term.

Average RMSE for put options using MLP is 113.918.

Table 5.7: RMSE for puts MLP

	short-term	mid-term	long-term
ITM	9.66	50.3	180.37
ATM	17.21	74.09	316.23
OTM	22.14	65.13	294.54

Average MAE for put options using MLP is 37.573.

Table 5.8: MAE for puts MLP

	short-term	mid-term	long-term
ITM	8.5	15.75	95.52
ATM	14.2	28.58	179.28
OTM	9.97	37.08	178.76

The above tables 5.7 and 5.8 summarises RMSE and MAE for put options prices using Multi-layer perceptron. The lowest RMSE is for short-term ITM options 9.66 and the highest for long-term ATM options 316.23. The lowest MAE is for short-term ITM options 8.5 and the highest for long-term ATM options 179.28. When we consider moneyness than ITM options have the lowest RMSE and MAE for all periods.

5.1.3 LSTM network

Average RMSE for call options using LSTM is 148.16.

Table 5.9: RMSE for calls LSTM

	short-term	mid-term	long-term
ITM	51.18	72.61	278.35
ATM	46.01	41.98	316.7
OTM	39.03	104.18	203.7

Average MAE for call options using LSTM is 58.95.

Table 5.10: MAE for calls BS

	short-term	mid-term	long-term
ITM	35.3	37.58	179.8
ATM	37.01	30.34	192.22
OTM	21.94	41.26	105.36

The tables 5.9 and 5.10 summarise RMSE and MAE for call options prices using LSTM network. The lowest RMSE is for short-term OTM options 39.03 and the highest for long-term ATM options 316.7. The lowest MAE is for short-term OTM options 21.94 and the highest for long-term ATM options 192.22. When we consider moneyness than OTM options have the lowest RMSE and MAE for short-term and long-term and ATM options have for mid-term.

Average RMSE for put options using LSTM is 135.471.

Table 5.11: RMSE for puts LSTM

	short-term	mid-term	long-term
ITM	23.12	49.75	279.15
ATM	18.94	70.97	392.75
OTM	35.17	63.79	217.28

Average MAE for put options using LSTM is 57.45.

Table 5.12: MAE for puts LSTM

	short-term	mid-term	long-term
ITM	20.59	22.03	159.97
ATM	16.31	24.29	214.07
OTM	25.21	38	103.62

The tables 5.11 and 5.12 summarise RMSE and MAE for put options prices using LSTM network. The lowest RMSE is for short-term ATM options 18.94 and the highest for long-term ATM options 392.75. The lowest MAE is for short-term ATM

options 16.31 and the highest for long-term ATM options 214.07. When we consider moneyness than ATM options have the lowest RMSE and MAE for short-term, ITM options for mid-term and OTM for long-term.

5.2 Realized volatility

5.2.1 Black-Scholes Model

Average RMSE for call options using BS is 160.433.

Table 5.13: RMSE for calls BS

	short-term	mid-term	long-term
ITM	47.25	85.32	314.87
ATM	5.05	45.51	392.64
OTM	31.64	113.35	234.23

Average MAE for call options using BS is 49.618.

Table 5.14: MAE for calls BS

	short-term	mid-term	long-term
ITM	16.51	39.21	157.03
ATM	2.99	28.68	223.1
OTM	6.46	40.97	104.62

The tables 5.13 and 5.14 summarise RMSE and MAE for call options prices using Black-Scholes formulas. The lowest RMSE is for short-term ATM options 5.05 and the highest for long-term ATM options 392.75. The lowest MAE is for short-term ATM options 2.99 and the highest for long-term ATM 223.1. When we consider moneyness than ATM options have the lowest RMSE and MAE for short-term and mid-term. OTM options have the lowest RMSE and MAE for long-term.

Average RMSE for put options using BS is 149.100.

Table 5.15: RMSE for puts BS

	short-term	mid-term	long-term
ITM	4.8	61.49	313.42
ATM	14.86	92.54	432.91
OTM	23.73	76.64	247.45

Average MAE for put options using BS 50.835.

Table 5.16: MAE for puts BS

	short-term	mid-term	long-term
ITM	3.83	34.12	176.25
ATM	11.38	58.82	263.48
OTM	13.47	53.06	145.38

The tables 5.15 and 5.16 summarise RMSE and MAE put options prices using Black-Scholes formulas. The lowest RMSE is for short-term ITM options 4.8 and the highest for long-term ATM options 432.91. The lowest MAE is for short-term ITM options 3.83 and the highest for long-term ATM options 263.48. When moneyness is considered than ITM options have the lowest RMSE and MAE for short-term and mid-term. OTM options have the lowest RMSE and MAE for long-term.

5.2.2 Multi-layer Perceptron

Average RMSE for call options using MLP is 169.88.

Table 5.17: RMSE for calls MLP

	short-term	mid-term	long-term
ITM	76.5	91.75	349.69
ATM	48.07	38.69	387.85
OTM	62.55	106.46	226.04

Average MAE for call options using MLP is 82.656.

Table 5.18: MAE for calls MLP

	short-term	mid-term	long-term
ITM	38.78	44.58	219.96
ATM	25.73	37.24	149.19
OTM	60.69	64.73	110.35

The tables 5.17 and 5.18 summarise RMSE and MAE for call options prices using Multi-layer perceptron. The lowest RMSE is for short-term ATM options 48.07 and the highest for long-term ATM options 387.85. The lowest MAE is for short-term ATM options 25.73 and the highest for long-term ITM options 219.96. When we consider moneyness than ATM options have the lowest RMSE and MAE for short-term and mid-term. OTM options have the lowest RMSE and MAE for long-term.

Average RMSE for put options using MLP is 104.022.

Table 5.19: RMSE for puts MLP

	short-term	mid-term	long-term
ITM	6.7	113.6	530.33
ATM	10.55	207.3	582.22
OTM	20.61	114.55	635.62

Average MAE for put options using MLP is 32.664.

Table 5.20: MAE for puts MLP

	short-term	mid-term	long-term
ITM	6.31	43.3	402.94
ATM	6.61	112.86	470.31
OTM	7.96	66.67	523.11

The tables 5.19 and 5.20 summarise RMSE and MAE for put options prices using Multi-layer perceptron. The lowest RMSE is for short-term ITM options 6.7 and the highest for long-term OTM options 635.62. The lowest MAE is for short-term

ITM options 6.31 and the highest for long-term OTM 523.11. When we consider moneyness than ITM options have the lowest RMSE and MAE for all periods.

5.2.3 LSTM network

Average RMSE for call options using LSTM is 148.16.

Table 5.21: RMSE for calls LSTM

	short-term	mid-term	long-term
ITM	51.18	72.61	278.35
ATM	46.01	41.98	316.7
OTM	39.03	104.18	203.7

Average MAE for call options using LSTM is 58.95.

Table 5.22: MAE for calls BS

	short-term	mid-term	long-term
ITM	35.3	37.58	179.8
ATM	37.01	30.34	192.22
OTM	21.94	41.26	105.36

The tables 5.21 and 5.22 summarise RMSE and MAE for call options prices using LSTM network. The lowest RMSE is for short-term OTM options 39.03 and the highest for long-term ATM options 316.7. The lowest MAE is for short-term OTM options 21.94 and the highest for long-term ATM options 192.22. When we consider moneyness than OTM options have the lowest RMSE and MAE for short-term and long-term. ATM options have the lowest RMSE and MAE for mid-term.

Average RMSE for put options using LSTM is 128.127.

Table 5.23: RMSE for puts LSTM

	short-term	mid-term	long-term
ITM	27.64	64.52	234.93
ATM	59.25	88.76	342.56
OTM	88.86	82.51	201.45

Average MAE for put options using LSTM is 75.510.

Table 5.24: MAE for puts LSTM

	short-term	mid-term	long-term
ITM	21.91	49.52	143.55
ATM	57.04	70.3	232.75
OTM	80.99	72.4	131.17

The tables 5.23 and 5.24 summarise RMSE and MAE for put options prices using LSTM network. The lowest RMSE is for short-term ITM options 27.64 and the highest for long-term ATM options 342.56. The lowest MAE is for short-term ITM options 21.91 and the highest for long-term ATM 232.75. When we consider moneyness than ITM option has the lowest RMSE and MAE for short-term and mid-term. OTM options have the lowest RMSE and MAE for long-term.

Conclusion

The goal of this thesis was to use recurrent neural network for option pricing and to compare them to traditional methods. To do so, we have created a way have to transform and fit option data to LSTM neural network using multivariate time series with different sequence lengths. The data we use are publicly available and able to use in real life situations.

We showed that our proposed method yields in some situation better results than traditional Black-Scholes model and Mulit-layer perceptron. However results we got do not allow to conclude that the method we proposed is simply better than Black-Scholes or MLP. It however allows to conclude that the method is viable and with further research it may yield interesting results. Moreover we have tried and we can conclude this for two different estimations of volatilities.

We believe that there are many potentially interesting improvements of the method we used for LSTM network training in this thesis. First of all, we can honestly state that hyperparameters we have found are sub-optimal. This is simply resulted by the fact that possible combinations of hyperparameters grows exponentially wit growing number of parameters and there are many parameters in LSTM network with one additional resulting from the transformation to tensor. Despite our best effort, we was only able to evaluate only few dozens of possible sets of hyperparameters. We were greatly limited by our hardware capabilities. Thus one can use more powerfull computer or cluster of computers for finding better hyperparameters.

Potentially interesting is to experiment with number timesteps in a sequence. We assumed that the more information we can get into the network, the better. However it is possible that for only last n sample are relevant for tomorrows option price and additional samples only import noise. It would be nice to gather more samples of different lengths and than training and predict only on sample with very similar timesteps. So if one would have only weekly options, than it would be reasonable to use let say last 5 samples or so.

Another area of improvement may be architecture of LSTM network. We have

used original architecture with default activation functions. It would be however interesting to experiment with different activation function and combination layers.

Further experimenting with different input variables may also yield interesting results. We wanted to use same inputs to there is no information asymmetry between method and there is theoretical support for them in Black-Scholes model. However further researchers may try added for instance volume or different time series to the variables already presented.

Another idea might be to use longer period of time as we did. We used 5 years because of our hardware limitations, but there much longer history available on mentioned resources.

In summary, we managed to successfully use LSTM networks for option pricing. Our results suggest that LSTM networks are not only viable way for option pricing, but potentially interesting way of pricing options. Additionally, we showed that neural networks are not that complicated as usually assumed and we the right package is used than it is easy to use very complex architectures such LSTM.

Bibliography

Anbazhagan, S. and N. Kumarappan (2013). "Day-Ahead Deregulated Electricity Market Price Forecasting Using Recurrent Neural Network". In: *IEEE Systems Journal* 7.4, pp. 866–872. ISSN: 1932-8184. DOI: 10.1109/JSYST.2012.2225733.

Andersen, Torben and Tim Bollerslev (1998). "Answering the Skeptics: Yes, Standard Volatility Models Do Provide Accurate Forecasts". In: *International Economic Review* 39.

Bakshi Gurdip Cao Charles, Chen Zhiwu (1997). "Empirical Performance of Alternative Option Pricing Models". In: *The Journal of Finance* 52.5, pp. 2003–2049. DOI: 10.1111/j.1540-6261.1997.tb02749.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1997.tb02749.x.

Bao, Wei, Jun Yue, and Yulei Rao (2017). "A deep learning framework for financial time series using stacked autoencoders and long-short term memory". In: *PLoS ONE* 12.

Baruníková, Michaela Vlasáková (2009). *Option Pricing: The empirical tests of the Black-Scholes pricing formula and the feed-forward networks*. Working Papers IES 2009/16. Charles University Prague, Faculty of Social Sciences, Institute of Economic Studies. URL: https://ideas.repec.org/p/fau/wpaper/wp2009_16.html.

Bengio, Y., P. Simard, and P. Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2, pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181.

Bergstra, James and Y Bengio (2012). "Random Search for Hyper-Parameter Optimization". In: *The Journal of Machine Learning Research* 13, pp. 281–305.

Bergstra, James et al. (2015). "Hyperopt: A Python library for model selection and hyperparameter optimization". In: *Computational Science & Discovery* 8.

Bernal, Armando, Sam Fok, and Rohit Pidaparthi (2012). *Financial Market Time Series Prediction with Recurrent Neural Networks*.

Black, Fischer and Myron Scholes (1973). "The Pricing of Options and Corporate Liabilities". In: *Journal of Political Economy* 81.3, pp. 637–654.

Claesen, Marc and Bart De Moor (2015). "Hyperparameter Search in Machine Learning". In: *CoRR* abs/1502.02127. arXiv: 1502.02127. URL: http://arxiv.org/abs/1502.02127.

Cybenko, George (1989). "Approximation by superpositions of a sigmoidal function. Math Cont Sig Syst (MCSS) 2:303-314". In: *Mathematics of Control, Signals, and Systems* 2, pp. 303–314.

Elman, Jeffrey L. (1990). "Finding structure in time". In: *Cognitive Science* 14.2, pp. 179 –211. ISSN: 0364-0213. DOI: https://doi.org/10.1016/0364-0213(90)90002-E. URL: http://www.sciencedirect.com/science/article/pii/036402139090002E.

Gers, Felix A., Jürgen A. Schmidhuber, and Fred A. Cummins (2000). "Learning to Forget: Continual Prediction with LSTM". In: *Neural Comput.* 12.10, pp. 2451–2471. ISSN: 0899-7667. DOI: 10.1162/089976600300015015. URL: http://dx.doi.org/10.1162/089976600300015015.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press. URL: http://www.deeplearningbook.org.

Graves, A. et al. (2009). "A Novel Connectionist System for Unconstrained Handwriting Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.5, pp. 855–868. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.137.

Hammer, Barbara (2001). "On the Approximation Capability of Recurrent Neural Networks". In: *Neurocomputing* 31.

Hastie, Trevor et al. (2004). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Math. Intell.

Herculano-Houzel, Suzana and Roberto Lent (2005). "Isotropic Fractionator: A Simple, Rapid Method for the Quantification of Total Cell and Neuron Numbers in the Brain". In: *Journal of Neuroscience* 25.10, pp. 2518–2521. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.4526-04.2005. eprint: http://www.jneurosci.org/content/25/10/2518.full.pdf. URL: http://www.jneurosci.org/content/25/10/2518.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-term Memory". In: *Neural computation* 9, pp. 1735–80.

Hopfield, J J (1982). "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the National Academy of Sciences* 79.8, pp. 2554–2558. ISSN: 0027-8424. DOI: 10.1073/pnas.79.8.2554. eprint:

http://www.pnas.org/content/79/8/2554.full.pdf. URL: http://www.pnas.org/content/79/8/2554.

Hornik, Kurt (1991). "Approximation Capabilities of Multilayer Feedforward Networks". In: *Neural Netw.* 4.2, pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: http://dx.doi.org/10.1016/0893-6080(91)90009-T.

Hull, John C. (2006). *Options, futures, and other derivatives*. 6. ed., Pearson internat. ed. Pearson Prentice Hall. XXII, 789. ISBN: 978-0-13-197705-1.

Hutchinson, James M., Andrew W. Lo, and Tomaso Poggio (1994). "A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks". In: Working Paper Series 4718. DOI: 10.3386/w4718. URL: http://www.nber.org/papers/w4718.

Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown (2011). "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Learning and Intelligent Optimization*, pp. 507–523.

Hyndman, Rob and Anne Koehler (2006). "Another look at measures of forecast accuracy". In: *International Journal of Forecasting* 22, pp. 679–688.

Hyndman, Rob J (2014). *Measuring forecast accuracy*. [Online; accessed 24-April-2018]. URL: pdfs.semanticscholar.org/af71/3d815a7caba8dff7248ecea05a5956b2a487.pdf.

International Settlement, Bank of (2018). *Global OTC derivatives market*. [Online; accessed 15-May-2018]. URL: https://www.bis.org/statistics/d5_1.pdf.

Itô, Kiyosi (1950). "Stochastic differential equations in a differentiable manifold". In: *Nagoya Math. J.* 1, pp. 35–47. URL: https://projecteuclid.org:443/euclid.nmj/1118764702.

Karpathy, Andrej (2018). *Introduction*. [Online; accessed 24-April-2018]. URL: http://cs231n.github.io/optimization-1/.

Kingma, Diederik and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. Vol. 25.

L. Samuel, Arthur (2000). "Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress". In: *IBM Journal of Research and Development* 3, pp. 206 –226.

Lecun, Yann et al. (2012). "Efficient BackProp". In: *Neural networks: Tricks of the trade*, pp. 9–48.

Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng (2013). "Rectifier nonlinearities improve neural network acoustic models". In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

Merton, Robert C. (1973). "Theory of Rational Option Pricing". In: *The Bell Journal of Economics and Management Science* 4.1, pp. 141–183. ISSN: 00058556. URL: http://www.jstor.org/stable/3003143.

Montréal Inc., Bourse de (2018a). *Overview*. [Online; accessed 24-April-2018]. URL: https://www.m-x.ca/qui_bref_en.php.

— (2018b). *S&P/TSX 60 Index Options (SXO)*. [Online; accessed 24-April-2018]. URL: https://www.m-x.ca/produits_indices_sxo_en.php.

Nitish Shirish Keskar Dheevatsa Mudigere, Jorge Nocedal Mikhail Smelyanskiy and Ping Tak Peter Tang (2016). "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *arXiv preprint arXiv:1609.04836*.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2012). "Understanding the exploding gradient problem". In: *CoRR* abs/1211.5063. arXiv: 1211.5063. URL: http://arxiv.org/abs/1211.5063.

Sak, Hasim, Andrew W. Senior, and Françoise Beaufays (2014). "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition". In: *CoRR* abs/1402.1128. arXiv: 1402.1128. URL: http://arxiv.org/abs/1402.1128.

Sonoda, Sho and Noboru Murata (2017). "Neural network with unbounded activation functions is universal approximator". In: *Applied and Computational Harmonic Analysis* 43.2, pp. 233 –268. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2015.12.005. URL: http://www.sciencedirect.com/science/article/pii/S1063520315001748.

V. Kondratenko, V and Yuri Kuprin (2003). "Using Recurrent Neural Networks To Forecasting of Forex". In:

White, Halbert (1989). "Some Asymptotic Results for Learning in Single Hidden-Layer Feedforward Network Models". In: *Journal of the American Statistical Association* 84.408, pp. 1003–1013. DOI: 10.1080/01621459.1989.10478865. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1989.10478865.

Widrow, B. and M. E. Hoff (1960). "Adaptive Switching Circuits". In: *1960 IRE WESCON Convention Record*. Reprinted in *Neurocomputing* MIT Press, 1988 ., pp. 96–104.

Witzany, J. (2013). *Financial Derivatives and Market Risk Management*. Oeconomica. ISBN: 9788024519807.

Witzany, Tomáš (2017). *Deep neural networks and their application for economic data processing*. Department of Theoretical Computer Science and Mathematical Logic, Charles University, Prague.

Y. Liu, Lily, Andrew J. Patton, and Kevin Sheppard (2012). "Does Anything Beat 5-Minute RV? A Comparison of Realized Measures Across Multiple Asset Classes". In: *Journal of Econometrics* 187.

List of Figures

List of Tables