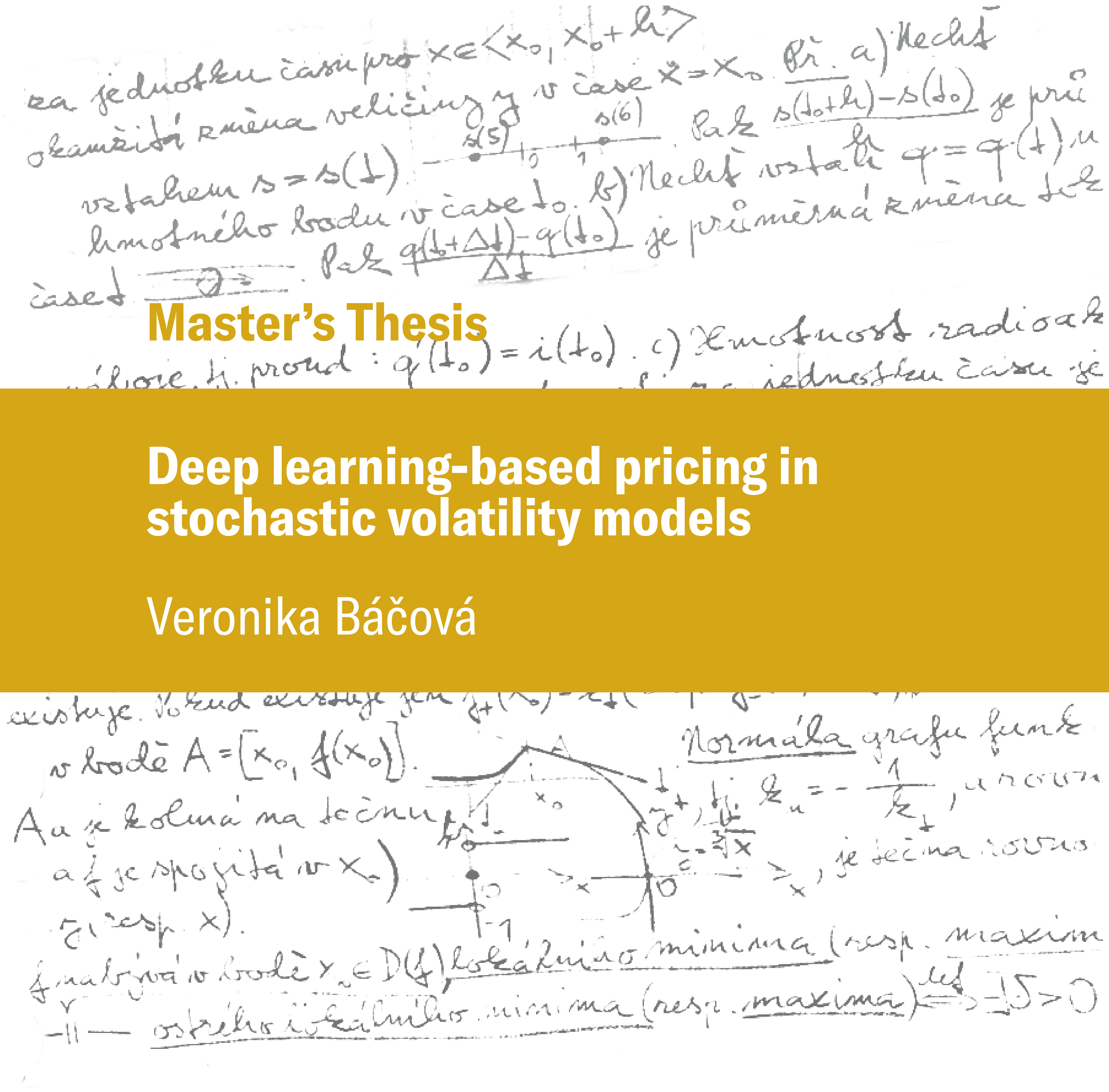
FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
MATHEMATICS



Master's Thesis

Deep learning-based pricing in stochastic volatility models

Veronika Báčová

PILSEN, CZECH REPUBLIC

2023

FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
MATHEMATICS

Master's Thesis

Deep learning-based pricing in stochastic volatility models

Bc. Veronika Báčová

Thesis advisor
Ing. Jan Pospíšil, Ph.D.

PILSEN, CZECH REPUBLIC 2023

Citation in the bibliography/reference list:
BÁČOVÁ, Veronika. *Deep learning-based pricing in stochastic volatility models*. Pilsen, Czech Republic, 2023. Master's Thesis. University of West Bohemia, Faculty of Applied Sciences, Department of Mathematics. Thesis advisor Ing. Jan Pospíšil, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Veronika BÁČOVÁ**
Osobní číslo:	**A21N0003P**
Studijní program:	**N0541A170005 Matematika a finanční studia**
Téma práce:	**Oceňování v modelech stochastické volatility založené na hlubokém učení**
Zadávající katedra:	**Katedra matematiky**

Zásady pro vypracování

1. Zpracujte rešerši obvykle používaných architektur hlubokých neuronových sítí pro oceňování v modelech stochastické volatility.
2. Popište a ve vhodném vývojovém prostředí implementujte (prediktivní) model na bázi hluboké neuronové sítě pro oceňování v modelech stochastické volatility, zejména v Hestonově modelu.
3. Analyzujte numerické vlastnosti modelu při použití různých hlubokých neuronových sítí a postupů jejich učení.
4. Srovnejte jednotlivé případy z pohledu úspěšnosti kalibrace na reálná tržní data, simulace a predikce.

Rozsah diplomové práce:	**40-80 stran**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování diplomové práce:	**tištěná**

Seznam doporučené literatury:

- Goodfellow, I., Y. Bengio, and A. Courville. Deep Learning. MIT Press, 2016.
- Horvath, B., Muguruza, A., and Tomas, M. (2021). Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models. Quant. Finance 21(1), 11-27, DOI: 10.1080/14697688.2020.1817974.
- Ruf, J. and Wang, W. (2020). Neural networks for option pricing and hedging: a literature review. J. Comput. Finance 24(1), 1-46, DOI 10.21314/JCF.2020.390.

Vedoucí diplomové práce:	**Ing. Jan Pospíšil, Ph.D.** Katedra matematiky
Datum zadání diplomové práce:	**3. října 2022**
Termín odevzdání diplomové práce:	**22. května 2023**

Západočeská univerzita v Plzni * Fakulta aplikovaných věd *

L.S.

[illegible] [illegible]

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Marek Brandner, Ph.D.
vedoucí katedry

V Plzni dne 3. října 2022

Declaration

I hereby declare that this Master's Thesis is completely my own work and that I used only the cited sources, literature, and other resources. This thesis has not been used to obtain another or the same academic degree.

I acknowledge that my thesis is subject to the rights and obligations arising from Act No. 121/2000 Coll., the Copyright Act as amended, in particular the fact that the University of West Bohemia has the right to conclude a licence agreement for the use of this thesis as a school work pursuant to Section 60(1) of the Copyright Act.

In Pilsen, on 19 May 2023

. . . .

Veronika Báčová

Acknowledgement

I would like to thank Ing. Jan Pospíšil, Ph.D. for his willingness, useful advice, and above all, for his patience and time devoted to consultations. My thanks also go to Ing. Kamil Ekštein, Ph.D. for his help with neural networks. Finally, I would like to thank my friends and family for their support and encouragement throughout my studies.

Abstract

This thesis is focused on option pricing in stochastic volatility models using neural networks. First, option prices in the Heston model are generated using the Heston-Lewis formula. A neural network is then trained using these prices to first estimate the parameters of the Heston model and then back-estimate option prices from these parameters. The trained neural network is also used to estimate option prices for real market data.

Abstrakt

Diplomová práce je zaměřena na oceňování opcí v modelech stochastické volatility pomocí neuronových sítí. Nejprve jsou vygenerovány ceny opcí v Hestonově modelu pomocí Heston-Lewisovy formule. Pomocí těchto cen je natrénovaná neuronová síť, která nejprve odhadne parametry Hestonova modelu a poté z těchto parametrů zpět odhadne ceny opcí. Natrénovaná neuronová síť je také použita na odhad cen opcí pro reálná tržní data.

Keywords

deep learning • neural networks • Heston model • option pricing

Contents

Glossary and Notation

$ADAM$	Adaptive moment estimation method
AI	Artificial intelligence
BS	Black Scholes model
DL	Deep learning
IV	Implied volatility
K	Strike price of an option
κ	Reversion rate
ML	Machine learning
NN	Neural network
r	Risk-free interest rate
$ReLU$	Rectified linear activation function
ρ	Correlation of the two Wiener processes
$S(t)$	Stock price process
T	Maturity time (in years)
θ	Long term variance
Θ	Parameter set of Heston model
$v(t)$	CIR process
$W(t)$	Wiener process

Introduction 1

Nowadays, the use of artificial intelligence is all around us. We can find artificial intelligence in web search engines, recommendation systems, self-driving cars, and so on. In this thesis, a neural network is used, which is one of the computational models used in artificial intelligence.

The beginning of artificial intelligence dates back to 1945, when mathematician Walter Pitts and neurophysiologist Warren McCulloch proposed the first model of artificial neurons [1]. Building on this idea, in 1949, Donald Hebb [2] demonstrated a rule for modifying the strength of connections between neurons (nowadays called Hebbian learning). The first neural network was created in 1958 by Frank Rosenblatt, who created an algorithm for pattern classification called the perceptron [3]. In 1962 Frank Rosenblatt found the backpropagation algorithm [4], which made it possible to train networks with multiple layers. In the following years, interest in neural networks decreased due to insufficient processing power. Interest in neural networks was renewed in 1986 when Rumelhart, Hinton, and Wilson in the book "Learning representations by back-propagating errors" [5] popularized the backpropagation algorithm, which is still widely used for training neural networks.

In the following years, there was an effort to find a theorem that would prove that a neural network can approximate any continuous function for inputs within a specific range. The first theorem was presented by George Cybenko in 1989 for the sigmoid activation function, which says that a neural network with just one hidden layer of N neurons, where $N \longrightarrow \infty$, can always approximate a multi-variant continuous function [6]. The Cybenko theorem had a historical impact in this field and the interest of other researchers in the field of universal approximation theorems increased. Similar results, with the difference that the activation function can be any bounded and non-constant function (e.g. ReLU), were reached in 1989 by K. Hornik, H. White, and M. Stinchcombe in paper "Multilayer Feedforward Networks are Universal Approximators" [7]. Hornik's theorem thus extends Cybenko's theorem and provides more freedom in the choice of activation functions for neural networks. The drawback of arbitrary-width universal approximation theorems (i.e. theorems considering a neural network with one hidden layer) is that the number of neurons in one hidden layer must be large to determine arbitrary accuracy. Another approach for defining the neural network architecture is the use of arbitrary-depth universal approximation theorems. The universal approximation theorem for width-bounded neural networks was proven in 2017 by Zhou Lu et al. [8]. To determine arbitrary accuracy using the arbitrary-depth universal approximation theorem, a large number of hidden layers are required. Therefore, in practice, limited depth and width are most commonly considered for constructing the architecture of neural networks.

At the beginning of the 20th century, an interest in option pricing also began. French mathematician Louis Bachelier was the first person who, in 1900, as a part of his doctoral thesis “The Theory of Speculation”, model the stochastic process now called the Wiener process (or Brownian motion) and its use for valuing stock options. In 1973, the authors F. Black and M. Scholes published the first work that became a cornerstone for option pricing [9]. One of the assumptions of the Black-Scholes model is the constant volatility of the underlying asset. However, volatility is not constant. That was observed, for example, during the stock market crash of October 1987. Therefore, models that considered volatility as a random process continued to emerge. For example, the Hull-White model in 1987 [10], Chesney and Scott model in 1989 [11], or Heston model in 1993 [12]. Although these models give better results than the Black Scholes model, their computation is time consuming and therefore neural networks are considered for option pricing. The use of neural networks for option pricing is presented, for example, in the work by J. Ruf and W. Wang [13].

The thesis is inspired by the paper "Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models" by B. Horvath, A. Muguruza, and M. Tomas [14]. The authors of this paper describe a consistent neural network based calibration method for a variety of volatility models that complete the calibration process for the entire implied volatility surface in milliseconds. One of these models is the Heston stochastic volatility model with five parameters, that are the inputs to the neural network. The output layer of the neural network contains 88 neurons, which is the size of a fixed grid of points to represent the implied volatility surface. Thus, the number of neurons in the input layer is smaller than the number of neurons in the output layer, which is not very common in practice. Furthermore, the paper assumes a zero risk-free interest rate. This thesis follows the opposite direction and considers 88 values at the input and 6 at the output (5 Heston parameters and the risk-free interest rate).

The thesis is structured in the following way. In Chapter 2, we introduce the basic definitions and principles that we will need in the rest of this thesis, such as the Black Scholes model, the Heston-Lewis model, the structure of a neural network, and the ADAM optimization method. At the end of the chapter, we present an example of applying a neural network to a function of two variables. In Chapter 3 the selection of unknown parameters using the Experiment Manager application in Matlab is introduced, and for optimal combination parameters, a neural network is trained. In Chapter 3, we also introduce data generation using the Heston-Lewis model and scaling for option pricing using a neural network, defining a neural network and training parameters using Experiment Manager, and the process of training a neural network. Adjustment of real data and numerical results for generated and real data are presented in Chapter 4. We conclude in Chapter 5, where our results are summarized and further possible research following this thesis is discussed.

Preliminaries 2

Preliminaries that will be used later in the text are briefly introduced in this section. We will follow the books written by B. Després [15], Y. B. Goodfellow, I. and A. Courville [1], A. L. Lewis [16], and S. E. Shreve [17]. We will also follow a paper written by F. Black and M. Scholes [9], B. Horvath, A. Muguruza, and M. Tomas. [14], S. Ruder [18], J. Ruf, and W. Wang [13]. In the first section are described the definitions of options, option payoff, forward, Wiener process, and geometric Wiener process. The following sections describe the Black-Scholes model, the Heston model, some basic information about neural networks (structure of neurons, hidden layers, activation functions, backpropagation, ADAM method), and a sample example.

2.1 Definitions

Let $(S(t), t \geq 0)$ be a price of an asset at time t. We will model the price as a stochastic (random) process.

Definition 2.1.1 (according to Section 4.5.2 in [17]) *An option is a contract between seller and buyer that gives the buyer the right (not the obligation) to sell or buy from the seller an underlying asset at a specific price (strike price K).*

Although there are several options available on the stock market, call and put options are the most common. A call option gives its holder the right to buy the underlying asset at a predetermined price (strike price K). Conversely, a put option gives its holder the right to sell. Options can be divided according to the positions. Long position means that the buyer of the option, its holder, has the right to exercise the option, and short position means that the seller is obliged to sell or buy the underlying asset at a predetermined price at the buyer's request.

An option contract is also characterized by its expiration (maturity time T) and exercise type. An American option allows the purchase or sale of the underlying asset during the entire time interval until expiration. A European option allows you to buy or sell only at the time of expiration.

Definition 2.1.2 (according to Section 4.5.2 in [17]) *Let K be a strike price at some future time T called maturity and $S(T)$ be the price of the underlying asset at exercise. Option payoff for European call option is $(S-K)^+$ and for European put option is $(K-S)^+$, where $(S-K)^+ = \max(S-K,0)$ and $(K-S)^+ = \max(K-S,0)$.*

If not stated otherwise, in the rest of the thesis, we will consider only European call options. Option payoffs for long/short call and long/short put are depicted in Figure 2.1.

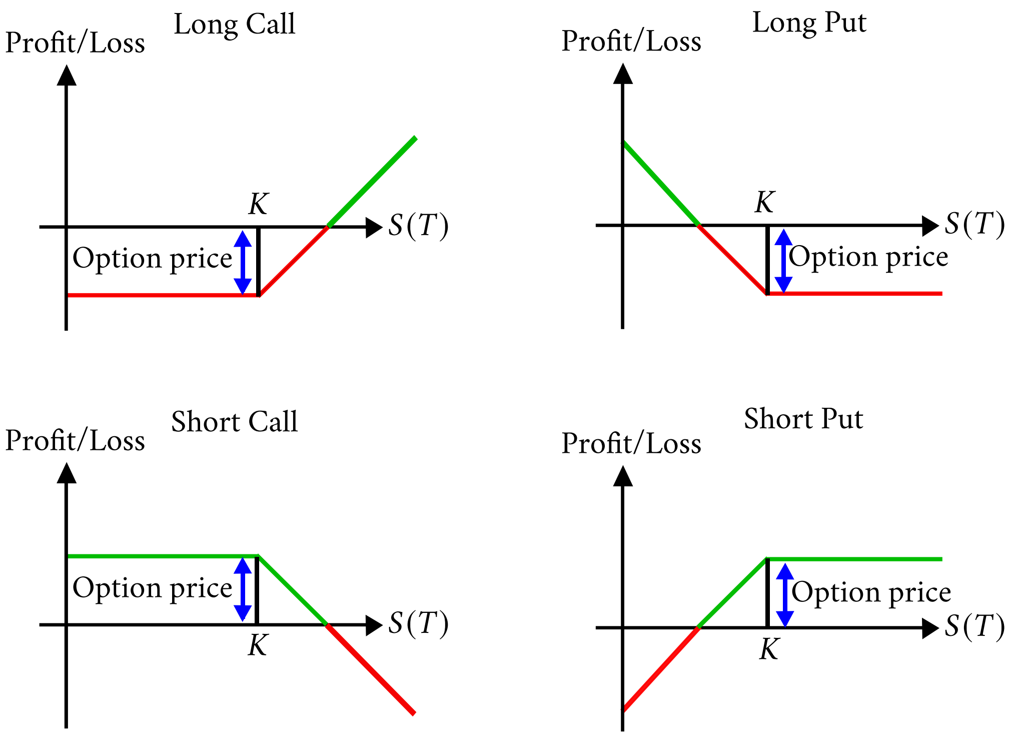


Figure 2.1: Option payoff for long/short call and long/short put option

Definition 2.1.3 (according to Section 5.6.1 in [17]) *A forward contract is an agreement to pay or sell a specified delivery price K (strike price) at a delivery date T (maturity time) for the asset whose price at time t is $S(t)$.*

Unlike an option, where the buyer has the right to buy or sell the underlying asset (at time T at price K), in a forward, the buyer has the obligation. The forward price formula $FW(t,T)$ of the asset without dividend payment at time t is (Section 5.6.1 in [17]):

$$FW(t,T) = S(t)e^{r(T-t)}. \tag{2.1}$$

Definition 2.1.4 (according to Section 3.3.1 in [17]) *Wiener process (Brownian motion) $(W(t),\ t \geq 0)$ is a random process with continuous trajectories that satisfies*

$W(0) = 0$ almost surely and for all $0 = t_0 < t_1 < ... < t_m$ the increments

$$W(t_1) = W(t_1) - W(t_0), W(t_2) - W(t_1), ..., W(t_m) - W(t_{m-1})$$

are independent and each of these increments is normally distributed with

$$E[W(t_{i+1}) - W(t_i)] = 0,$$

$$Var[W(t_{i+1}) - W(t_i)] = t_{i+1} - t_i.$$

Definition 2.1.5 (according to Section 3.4.3 in [17]) *Let α and $\sigma > 0$ be constants. A stochastic process $S(t)$ is said to follow a Geometric Wiener process (Brownian motion) if it satisfies*

$$\mathrm{d}S(t) = \alpha S(t)\,\mathrm{d}t + \sigma S(t)\,\mathrm{d}W(t)$$

with the analytic solution

$$S(t) = S(0)\exp\left\{\sigma W(t) + (\alpha - \tfrac{1}{2}\sigma^2)t\right\}.$$

The Geometric Wiener process is the asset-price model used in the Black-Scholes option-pricing model.

2.2 Black Scholes model

The Black-Scholes model [9] is used to value assets that follow a geometric Wiener process with constant volatility. The model was constructed by economists Black Fischer and Myron Scholes and published in 1973. The model builds on earlier research by Edward O. Thorpe, Paul Samuelson, and Robert C. Merton [19]. Merton and Scholes got the Nobel Prize in Economics (in 1997) for this model and related work[1]. Black Fischer did not live to receive the Nobel Prize, he died in 1995.

Assumptions of Black Scholes model are (Section 3.4.3 in [17]):

- the stock does not pay a dividend,
- markets are random,
- there are no transaction costs or taxes in buying the option,
- the price of the underlying asset is geometric Wiener process with known and constant risk-free rate μ and volatility σ,

[1]precisely "for a new method to determine the value of derivatives", `https://www.nobelprize.org/prizes/economic-sciences/1997/press-release/`

- the option can be exercised only at expiration T (European option) and
- there is no possibility of arbitrage.

The Black-Scholes partial differential equation is (Section 4.5.3 in [17]):

$$\frac{\partial}{\partial t}c(t,x) + rx\frac{\partial}{\partial x}c(t,x) + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2}{\partial x^2}c(t,x) = rc(t,x), \tag{2.2}$$

where

$c(t,x)$	is the price of the option as a function of time t and stock price x,
σ	is volatility of stock and
r	is risk-free interest rate.

The solution of the partial differential equation (2.2), Black-Scholes option-pricing formula, for the European call option is (Section 4.5.4 in [17]):

$$C(t,x) = S(t)\Phi(d_1) - K\mathrm{e}^{-r(T-t)}\Phi(d_2), \tag{2.3}$$

where

Φ	is the cumulative distribution function of normal distribution,
$\sigma > 0$	is the volatility parameter,
T	is the time to maturity (in years),
r	is the risk-free interest rate,
K	is the strike price,
$S(t)$	is the current stock price and

$$d_1 = \frac{\ln\frac{S(t)}{K} + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}},$$

$$d_2 = d_1 - \sigma\sqrt{T-t}.$$

The parameter σ for which the theoretical option price given by equation (2.3) agrees with the market price is called implied volatility σ_{IV} and cannot be solved analytically. If the implied volatility is plotted against strike prices with the same expiration date, a generally convex strike price function is obtained. This phenomenon is referred to as a volatility smile (or a volatility skew).

The Black-Scholes model assumes asset valuations that follow a geometric Wiener process with constant volatility. However, in the markets, the volatility of the asset is not constant. One possibility is to model volatility as a stochastic process and we speak about stochastic volatility models.

2.3 Heston model

The Heston model is a stochastic volatility model that assumes that the volatility of an asset is not constant but follows the mean-reverting [20] stochastic process (CIR). The model is named after Steven L. Heston, an American financier, mathematician and economist.

The Heston model dynamics reads (Section 6.9.7 in [17]):

$$\begin{aligned} \mathrm{d}S(t) &= rS(t)\,\mathrm{d}t + \sqrt{v(t)}S(t)\,\mathrm{d}W^s(t) \\ \mathrm{d}v(t) &= -\varkappa(v(t)-\theta)\,\mathrm{d}t + \sigma\sqrt{v(t)}\,\mathrm{d}W^v(t) \\ \mathrm{d}W^s(t)\,\mathrm{d}W^v(t) &= \rho\,\mathrm{d}t, \end{aligned} \tag{2.4}$$

where

$W^s(t)$ and $W^v(t)$	are Wiener processes,
ρ	is the correlation of the two Wiener processes,
θ	is the long term variance,
$\varkappa$	is the reversion rate,
σ	is the volatility of volatility parameter,
$v(t)$	is the volatility process and
r	is the risk-free interest rate.

The CIR process cannot reach negative values if the Feller's condition

$$2\varkappa\theta \geq \sigma^2 \tag{2.5}$$

is satisfied.

Heston derived in [12] a semi-closed formula. The most accurate formula used nowadays is the formula by Lewis ([16], Chapter 2) who used

$$C(t,S(t),K) = S(t) - K\mathrm{e}^{-r_f(T-t)}\frac{1}{\pi}\int_{0+i/2}^{+\infty+i/2} \mathrm{e}^{-ikX}\frac{H}{k^2-ik}\,\mathrm{d}k, \tag{2.6}$$

where $X = \ln S(t)/K + r(T-t)$ and

$$H = \exp\left(\frac{2\varkappa\Theta}{\sigma^2}\left[qg - \ln\frac{1-he^{-\xi q}}{1-h}\right] + vg\left(\frac{1-\mathrm{e}^{-\xi q}}{1-he^{-\xi q}}\right)\right),$$

where

$$g = \frac{b-\xi}{2},\ h = \frac{b-\xi}{b+\xi},\ q = \frac{\sigma^2(T-t)}{2},$$

$$\xi = \sqrt{b^2 + \frac{4(k^2-ik)}{\sigma^2}},$$

$$b = \frac{2}{\sigma^2}(ik\rho\sigma + \varkappa).$$

Although the Heston-Lewis formula better describes the behaviour of the underlying asset, the calculation time for a large number of options might be time-consuming, therefore an approximation by neural networks is a promising approach to valuing the option price reliably and fast.

2.4 Neural network

A neural network is one of the computational models used in artificial intelligence that is inspired by the biological neuron. Units, also called artificial neurons, are connected by edges (in biological brain synaptic connections), that allow the signal to be transmitted to other neurons. Every neuron has a bias, and every edge has a weight that determines the strength of the signal (for more information, see Section 2.4.1). Neurons are most often grouped into layers, that can make transformations on their inputs. In this thesis, we will consider feed-forward neural networks, where signals move through the layers from the input layer through hidden layers to the output layer. For numerical reasons, the input values are often transformed to the interval $[-1, 1]$. Hidden layers require choosing the activation function that will be used to compute the hidden layer values. The basic structure of the neural network with three fully connected hidden layers, an input layer with 88 neurons, and an output layer with 6 neurons is depicted in Figure 2.2.

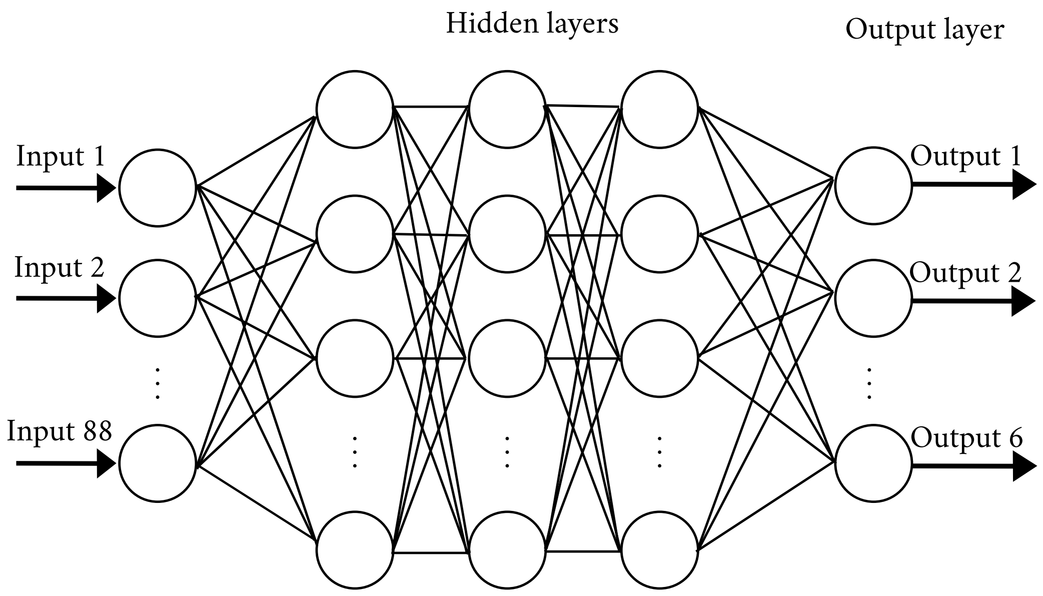


Figure 2.2: An example of neural network structure, an input layer, three fully connected hidden layers and an output layer

2.4.1 Structure of a neuron

Each neuron receives inputs x_i with weights w_i and a bias b. The so called net inputs are calculated as

$$z = \sum_i x_i w_i + b.$$

Output of the neuron is then given by the activation function f

$$y = f(z).$$

Neuron structure is depicted in Figure 2.3, where $x_1, \dots, x_n$ are the neurons in the previous layer, $w_1, \dots, w_n$ are the weights entering the neuron, b is the bias, f is the chosen activation function, and y is the output of the neuron.

There are many types of activation functions (see also [15], Chapter 2), e.g. ReLU (rectified linear activation function)

$$f(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases},$$

which is far more computationally efficient when compared to the hyperbolic tangent (tanh) or sigmoid function and in many applications it leads to similar or better results. The disadvantage of the ReLU function is that it can cause a “dead ReLU” problem, which occurs when the input to the activation function is negative. In this case, the ReLU function returns zero, and thus the gradient is zero and the weights are not updated. The neural network stops learning and becomes dead. Once the network becomes dead, there is no way to recover the network to learn.

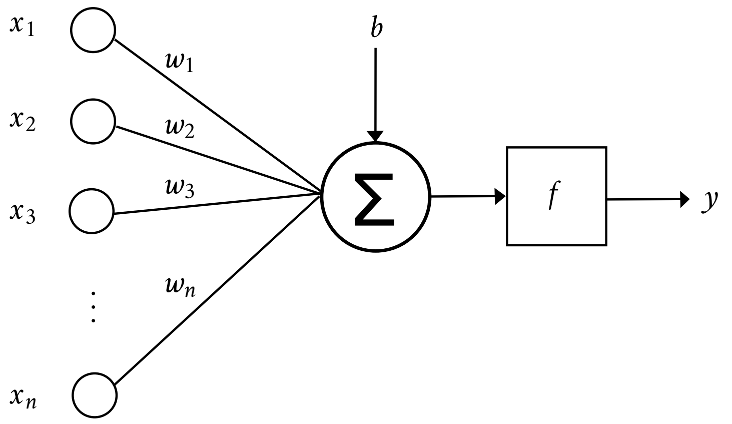


Figure 2.3: Neuron structure

To overcome the “dead ReLU” problem, a simple modification called Leaky ReLU activation function (LReLU) may be considered. It is defined as

$$f(z) = \begin{cases} z & \text{if } z \leq 0 \\ \alpha z & \text{otherwise} \end{cases}, \tag{2.7}$$

where α is a small slope parameter. Due to the non-zero parameter α, Leaky ReLU does not cause the “dead ReLU” problem and is also far more computationally efficient when compared to the hyperbolic tangent or sigmoid function. Leaky ReLU is used in this thesis to train the neural network in Example 1 (see Chapter 2.4.4). ReLU activation function is depicted in Figure 2.4 and Leaky ReLU activation function with slope parameter $\alpha = 0.2$ is in Figure 2.5.

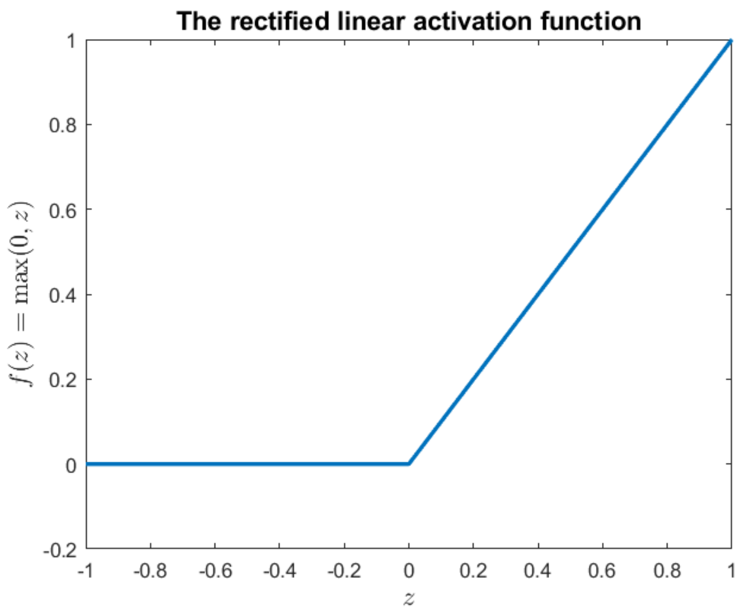


Figure 2.4: The rectified linear activation function on the interval $[-1, 1]$

To train the neural network for option pricing, a sigmoid function is considered as the activation function based on the analysis in Chapter 3.4. The sigmoid function is defined as

$$f(z) = \frac{1}{1 + \mathrm{e}^{-z}}$$

and depicted in Figure 2.6.

2.4.2 Backpropagation

Backpropagation is one of the most commonly used algorithms for training feed-forward neural networks found in 1962 by Frank Rosenblatt [4]. The backpropagation algorithm changes the biases and weights to minimize the objective function.

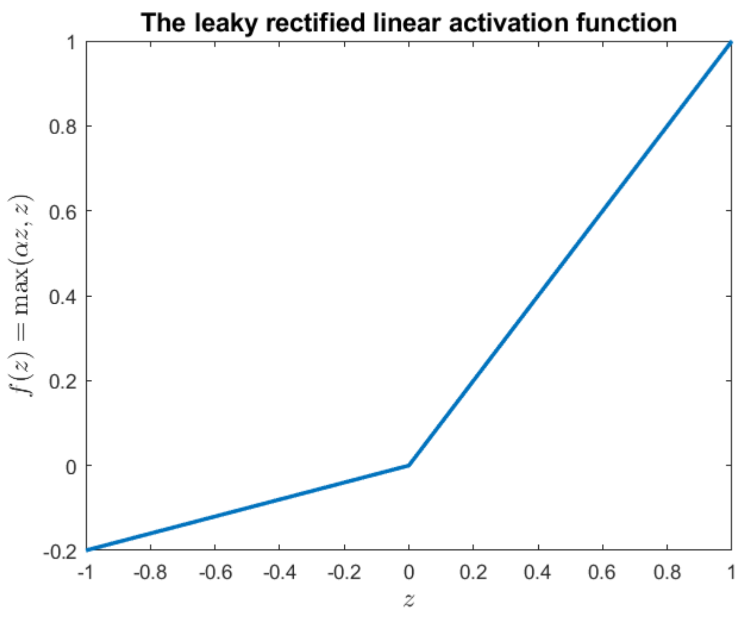


Figure 2.5: The leaky rectified linear activation function on the interval $[-1, 1]$

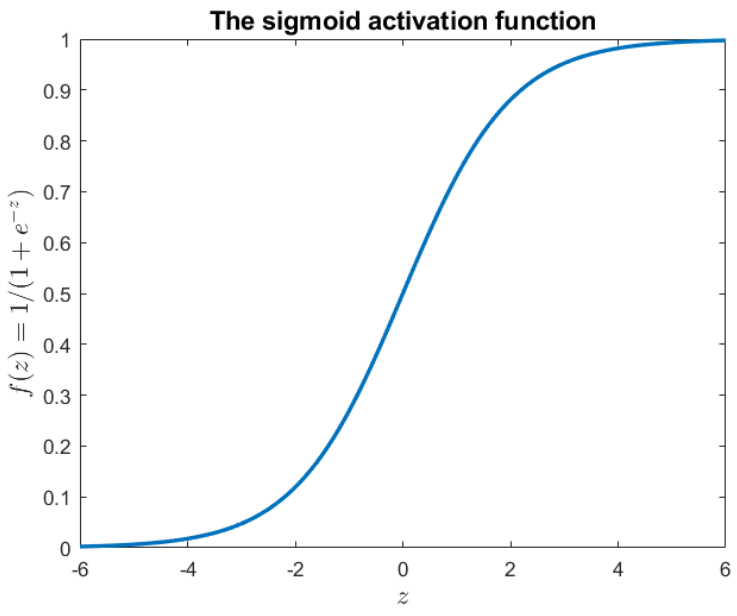


Figure 2.6: The sigmoid activation function on the interval $[-6, 6]$

The objective function can be defined as

$$J(\Theta) = \sum_{(x,y)\in D} \varphi_y(f(x, \Theta)),$$

where φ_y is some cost function. Typically we consider the nonlinear least squares problem with

$$J(\Theta) = \sum_{(x,y)\in D} |y - f(x,\Theta)|^2,$$

where y and $f(x,\Theta)$ are our observed values and values predicted using a neural network respectively. To find the minimum of a function $J(\Theta)$, it is necessary to find the gradient and take a step in the opposite direction (steepest descent). For this, we can use the chain rule, which is a mathematical procedure for calculating the derivative of a composite function. The discrete form of the gradient descent method can be written as

$$\Theta_{n+1} = \Theta_n - \alpha \nabla J(\Theta_n), \tag{2.8}$$

where α is called a learning rate (LR) and $\nabla J(\Theta_n)$ is calculated using the chain rule.

Theorem 2.4.1 (according to Section 5.1 in [15]) *Let $J : \Omega \longrightarrow \mathbb{R}$ and $\sigma > 0$ be a continuous function over a closed bounded set $\Omega \subset \mathbb{R}^q$ in finite dimension. Then there exists a minimizer $\Theta^* \in \Omega$ such that $J(\Theta^*) \leq J(\Theta)$ for all $\Theta \subset \Omega$.*

There exist many sophisticated and efficient optimization algorithms. One extension of the gradient descent optimization algorithm is a momentum method

$$\begin{aligned} v_t &= \gamma v_{t-1} + \alpha \nabla J(\Theta) \\ \Theta_{n+1} &= \Theta_n - v_t, \end{aligned} \tag{2.9}$$

where γ is coefficient of momentum (see [18]).

Figure 2.7 depicts a comparison of momentum and the gradient descent algorithm. The upper left part of Figure 2.7 shows the gradient descent algorithm, where the black lines indicate iterations given by expression (2.8). The upper right part shows the momentum method, where the black lines are the gradient descent steps, the orange line is a momentum step, and the blue line is an actual step, i.e. iteration given by expression (2.9). The bottom part of Figure 2.7 shows the progress of the gradient descent algorithm (on the left) and momentum (on the right) on a convex function.

Other extensions of the gradient descent algorithm include Nesterov's accelerated gradient descent, Adagrad, Adadelta (for more, see [18]). In this thesis, we consider the ADAM method described in Chapter 2.4.3.

2.4.3 ADAM method

One of the methods that modify the weights and learning rates of neural networks in order to minimize losses is the Adaptive moment estimation method (ADAM) (for

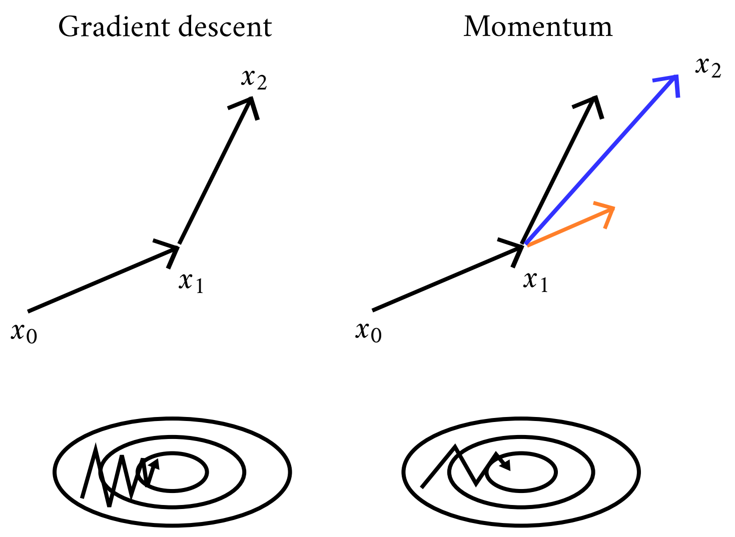


Figure 2.7: Comparing momentum and gradient descent algorithm

more information about other similar methods, we link the reader to [18]). Adam keeps an exponentially decaying average of past gradients m_t, similar to momentum (Section 4.6 in [18]):

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \end{aligned}$$

where m_t is an estimate of the first moment vector, v_t is an estimate of the second moment vector (m_0 and v_0 are initialized as zero vectors), β_1 and β_2 are ADAM method parameters, and g_t is a gradient of weights and biases. The authors of the ADAM method observe that m_t and v_t are biased towards zero. The first and second moment estimations after bias correction are (Section 4.6 in [18]):

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

and formula for updating the parameter θ (weights and biases) based on formula (2.8) is

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}},$$

where α is the learning rate (LR) and ϵ is a small constant for preventing divide-by-zero errors (both are also ADAM method parameters).

Input parameters cannot be chosen arbitrarily, for example β_1 and β_2 must satisfy the criterion (Remark 5.2.7 in [15])

$$4(\beta_1 - 1) \leq \beta_2 - 1. \tag{2.10}$$

It's also important to choose the right learning rate value α. The loss function for different values of the learning rate is shown in Figure 2.8. If the chosen learning rate is too small, the convergence may be slow, and conversely, if the learning rate is too large, the loss function may fluctuate around the minimum or even diverge [21].

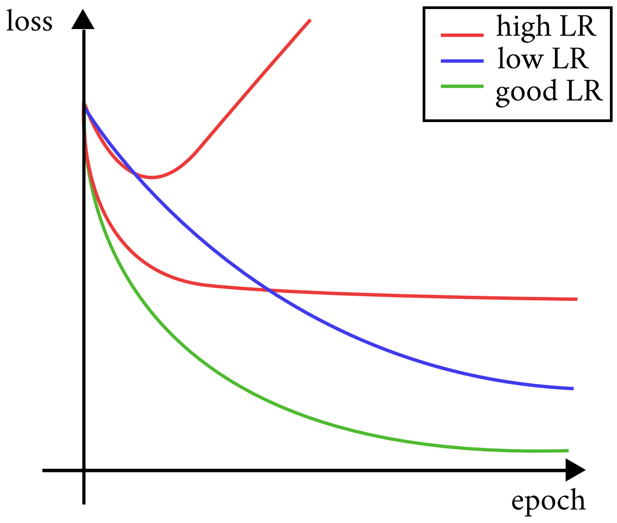


Figure 2.8: Effects of different learning rates

2.4.4 Example 1

Let us now consider the function

$$f(x, y) = \sqrt{x^2 + y^2}, \tag{2.11}$$

which is the calculation of the hypotenuse in a right triangle using the Pythagorean theorem. Data are randomly generated on the interval $[0, 1]$ using a function (2.11), saved in **trainset_hypotenuse.csv** and pictured in Figure 2.9.

The input dataset is split into training (5/6) and testing (1/6). Then, a neural network is created with two inputs (x and y), two hidden layers, each with 32 neurons, and one output. Leaky ReLU is chosen as the activation function (see (2.7)) and the adaptive moment estimation method (ADAM) is chosen as the optimization method (more information about method in Chapter 2.4.3) with parameters $\beta_1 = 0.99$, $\beta_2 = 0.99$ (which meet the condition 2.10), $\alpha = 0.01$ and $\epsilon = 1 \times 10^{-8}$. The number of epochs is set to 1 000 (how many times the network should go through the entire dataset during training) and the minibatch size is set to 50 (sample size, i.e. how much data is trained in one iteration). The neural network is trained with the ADAM optimizer and stored in **experiment.mat**.

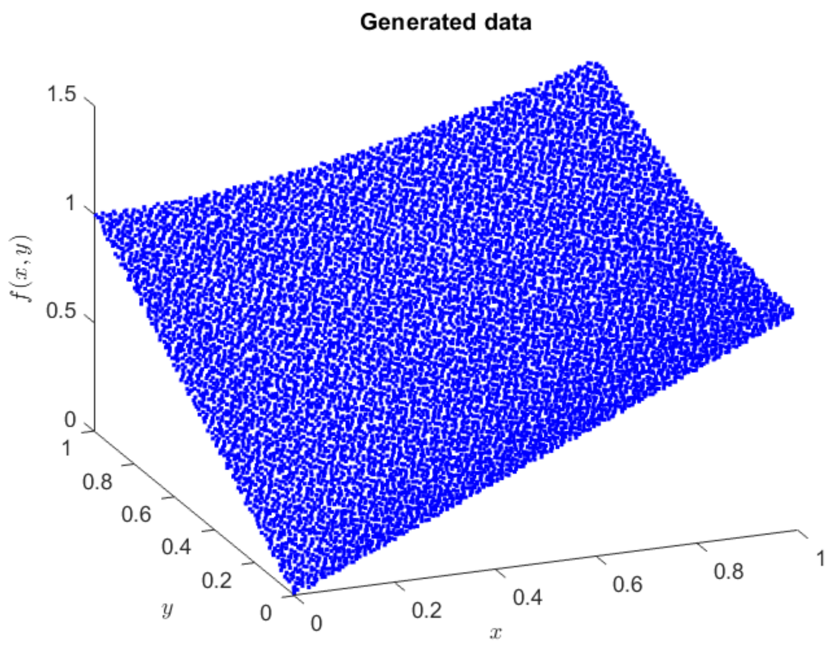


Figure 2.9: Generated data for function $f(x, y) = \sqrt{x^2 + y^2}$.

2.4.5 Universal approximation theorem

Universal approximation theorems are theorems that prove that, given certain conditions, a neural network can approximate any continuous function. Most universal approximation theorems can be divided into two classes. The first class, called "arbitrary width", considers a single hidden layer with an arbitrary number of neurons [6] and the second class, called "arbitrary depth", focuses on the case of a limited number of neurons, and an arbitrary number of hidden layers [7].

Example (Cybenko Theorem): *The Cybenko Theorem is verified by a shallow neural network with two-dimensional inputs and one-dimensional output. Data used for verification are also generated in Example* 1 *(more in Chapter 2.4.4). The sigmoid function is chosen as the activation function, and as an optimizer method, ADAM is used. Parameters are* $\beta_1 = 0.9$, $\beta_2 = 0.99$ *(which meet the condition (2.10)),* $\alpha = 0.001$ *and* $\epsilon = 1 \times 10^{-8}$*. The number of neurons in the hidden layer ranges between* 100 *and* 1000*. With an increasing number of neurons, the final loss should decrease, which is numerically verified in Figure 2.10, where the final loss after* 10 *epochs is shown.*

Example (Hornik Theorem): *Data used for verification are generated in Example* 1 *(more in Chapter 2.4.4). As the activation function, ReLU function is chosen and as an optimizer method, ADAM is used. Parameters are* $\beta_1 = 0.9$, $\beta_2 = 0.999$ *(which meet the condition (2.10)),* $\alpha = 0.001$ *and* $\epsilon = 1 \times 10^{-8}$*. The number of neurons in the hidden layer ranges between* 100 *and* 1000*. Figure 2.11 shows that with an increasing* *number of neurons, the final loss (after 10 epochs) decreases.*

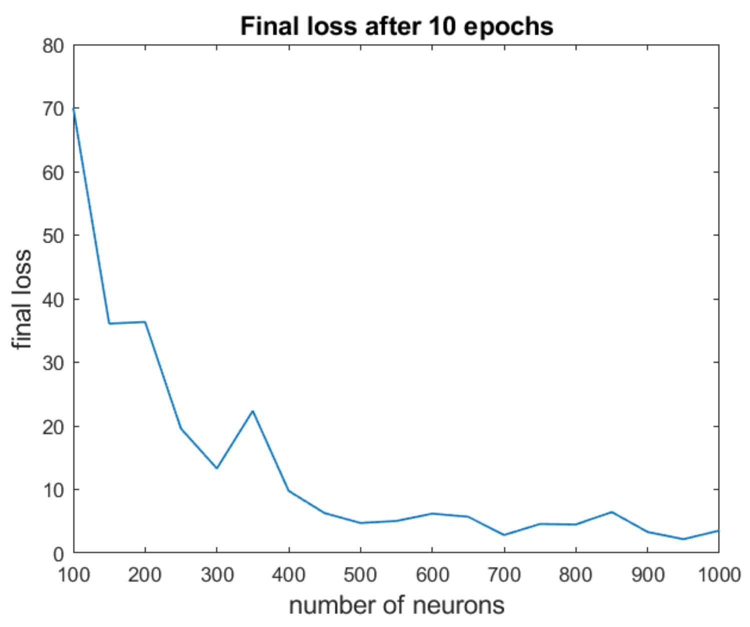


Figure 2.10: Verification of Cybenko Theorem

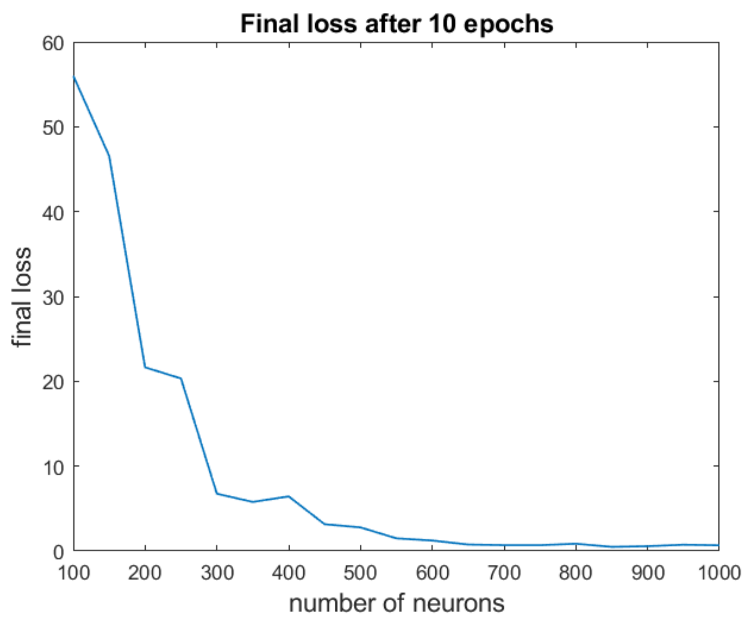


Figure 2.11: Verification of Hornik Theorem

Methodology 3

This chapter describes how to select the most appropriate parameters using the Experiment Manager App[1]. The application Experiment Manager is part of the Deep Learning toolbox in Matlab and allows the training of a neural network for combinations of unknown parameters in parallel within a given range. To run the Experiment Manager, it is necessary to select a range of unknown parameters and create a function in Matlab that will be run with all parameter combinations. The resulting table shows how well the neural network can be section trained for each parameter choice (training/validation loss).

In the first section, for Example 1 (from Section 2.4.4), the selection of optimal parameters is first described, followed by training a neural network with these optimal parameters and comparing the results obtained by the trained neural network with the test data set. The next section describes the generation of option prices using the Heston model (described in Section 2.3) and the calculation of implied volatility. Data that do not satisfy the Feller condition, contain arbitrage, and do not satisfy the convex function for the implied volatility surface are removed from the generated option price data. These remaining data are then scaled and stored. The third section describes a neural network architecture for option prices and uses Experiment Manager to find the combination of parameters (neurons in hidden layers) with the smallest error. The last section describes the training of the neural network, the selection of the activation function, and the parameters for training the network using the ADAM method.

3.1 Example 1 revisited

For Example 1 (see Section 2.4.4), the Matlab application Experiment Manager was used to estimate the optimal network and parameter combination. The Experiment Manager application returns a table containing the traning/validation root mean

[1]`https://www.mathworks.com/help/deeplearning/experiment-manager.html` [cit. on 8 May 2023]

square error (RMSE)

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

where y_i is the i-th validation data and $\hat{y}_i$ is the i-th predicted data, and traning/validation loss typically

$$\mathrm{loss} = \frac{\mathrm{RMSE}^2}{2} \tag{3.1}$$

for all parameter combinations. The Experiment Manager finds the combination of parameters that minimizes validation loss given by the expression (3.1).

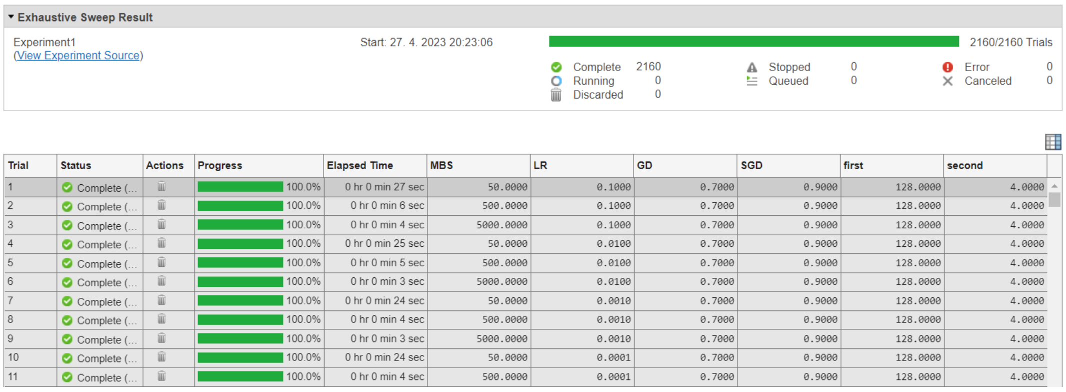
We consider a neural network with a ReLU activation function, the ADAM optimization method, and two hidden layers (based on Cybenko's Theorem, see Chapter 2.4.5, the number of hidden layers should be sufficient). The number of neurons in each hidden layer is chosen as the unknown parameter of the network. The other unknown parameters are β_1 and β_2 (input parameters of the ADAM method), size of minibatch (sample size) and α (initial learning rate). The values of the network training parameters are shown in Table 3.1. The number of neurons in the hidden layers is chosen as a power of 2 (the number of network input values) and the minibatch size is chosen as the number of training data (10 000) divided by 200, 20 and 2.

Table 3.1: Selection of parameter values for neural network training using Experiment Manager

	parameter values
number of neurons in the first hidden layer	$[128, 256, 512]$
number of neurons in the second hidden layer	$[4, 8, 16, 32]$
β_1	$[0.7, 0.8, 0.9, 0.99, 0.999]$
β_2	$[0.9, 0.99, 0.999]$
minibatch size	$[50, 500, 5\,000]$
α	$[0.1, 0.01, 0.001, 0.0001]$

For parameter values in Table 3.1 neural networks are trained in Experiment Manager, and the training process is shown in Figure 3.1. Combination of parameters with the smallest validation loss (given by the expression (3.1)) is $\beta_1 = 0.99$, $\beta_2 = 0.99$, minibatch size $= 50$ and $\alpha = 0.01$. Number of neurons in the first and the second hidden layers is 512 and 4 respectively.

For this optimal combination of parameters, a neural network is trained and a prediction is made on the test data. We calculate the minimum, mean, and maximum value of the difference between the predicted values and the test values, which is 4.54×10^{-9}, 2.43×10^{-5} and 1.99×10^{-4} respectively.



Exhaustive Sweep Result

Experiment1
(View Experiment Source)

Start: 27. 4. 2023 20:23:06

2160/2160 Trials

Complete	2160	Stopped	0	Error	0		
Running	0	Queued	0	Canceled	0		
Discarded	0						

Trial	Status	Actions	Progress	Elapsed Time	MBS	LR	GD	SGD	first	second
1	✓ Complete (…		100.0%	0 hr 0 min 27 sec	50.0000	0.1000	0.7000	0.9000	128.0000	4.0000
2	✓ Complete (…		100.0%	0 hr 0 min 6 sec	500.0000	0.1000	0.7000	0.9000	128.0000	4.0000
3	✓ Complete (…		100.0%	0 hr 0 min 4 sec	5000.0000	0.1000	0.7000	0.9000	128.0000	4.0000
4	✓ Complete (…		100.0%	0 hr 0 min 25 sec	50.0000	0.0100	0.7000	0.9000	128.0000	4.0000
5	✓ Complete (…		100.0%	0 hr 0 min 5 sec	500.0000	0.0100	0.7000	0.9000	128.0000	4.0000
6	✓ Complete (…		100.0%	0 hr 0 min 3 sec	5000.0000	0.0100	0.7000	0.9000	128.0000	4.0000
7	✓ Complete (…		100.0%	0 hr 0 min 24 sec	50.0000	0.0010	0.7000	0.9000	128.0000	4.0000
8	✓ Complete (…		100.0%	0 hr 0 min 4 sec	500.0000	0.0010	0.7000	0.9000	128.0000	4.0000
9	✓ Complete (…		100.0%	0 hr 0 min 3 sec	5000.0000	0.0010	0.7000	0.9000	128.0000	4.0000
10	✓ Complete (…		100.0%	0 hr 0 min 24 sec	50.0000	0.0001	0.7000	0.9000	128.0000	4.0000
11	✓ Complete (…		100.0%	0 hr 0 min 4 sec	500.0000	0.0001	0.7000	0.9000	128.0000	4.0000

Figure 3.1: Training neural networks using the Experiment Manager

The error histogram for the test data and the predicted data using the neural network is depicted in Figure 3.2.

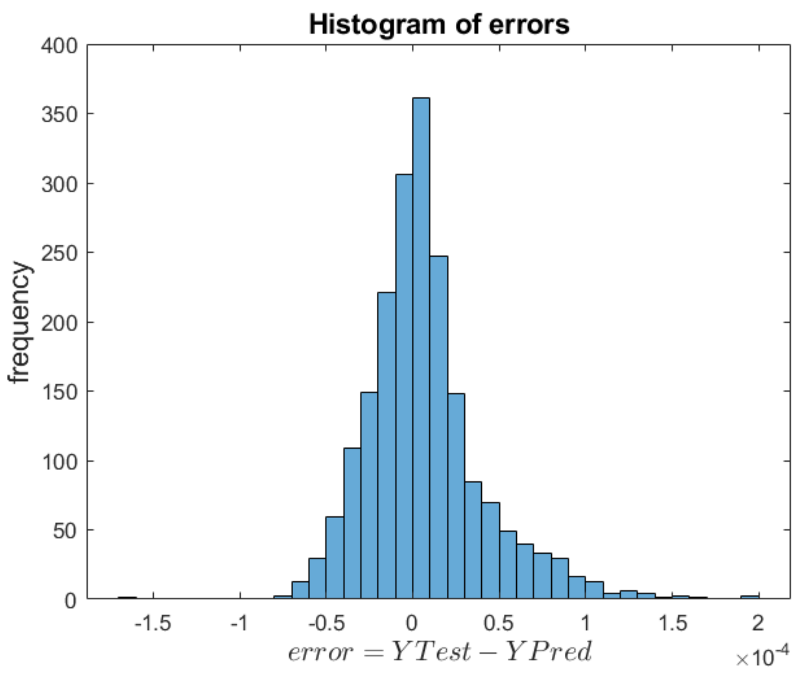


Figure 3.2: Histogram of the difference between the predicted values and the test values

3.2 Synthetic data generating

In this section, we describe how we generated synthetic data for NN training purposes. To generate option prices, the ranges of the option, market and Heston model parameters are first defined. The option parameters are maturities and moneyness. Moneyness is a dimensionless quantity defined here as $m = \frac{K}{S0}$. Maturities are chosen as $[0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0]$ and moneyness are from 0.5 to 1.5 with increments of 0.1. Market parameter is the interest rate with ranges from 0 to 0.1. Heston model parameter ranges are in Table 3.2.

Table 3.2: Heston model parameter ranges for synthetic option prices generator

	parameter ranges
$v(0)$	$[0.00001, 1]$
κ	$[0.0001, 50]$
θ	$[0.0001, 1]$
σ	$[0.0001, 4]$
ρ	$[-1, 1]$

We randomly (low discrepancy Sobol sequence is used) generate 100 000 combinations of parameters in the given ranges. For these parameters and with the spot price (stock price at the time 0) considered equal to 1 is by Heston-Lewis model (for more, see Chapter 2.3) count option price. To calculate the implied volatility, we use a function **myblsimpv.m** that evaluates the difference between the price calculated using the Heston-Lewis formula and the Black Scholes model with an unknown parameter σ. Using nonlinear least squares, the optimal fit of σ is found, which is called implied volatility. An example of prices and implied volatilities for a grid of different moneyness and maturities are shown in Figure 3.3. The parameters for the Heston model, prices and implied volatilities are stored in the files **trainset_prices.csv** and **trainset_ivols.csv**.

3.2.1 Data cleanup

Heston model parameters must satisfy the Feller's condition (2.5). This condition is not satisfied in 18 664 cases out of 100 000 randomly generated samples. Moreover, we also clean those option price surfaces that do not satisfy the no-arbitrage conditions in particular the vertical, butterfly and calendar spreads arbitrage conditions are tested. More precisely, in a grid of call option prices $C_{i,j}$ containing different strikes K_i (as mentioned before, $S(0) = 1$ is considered, so $K_i = m_i$) and maturities T_j, we calculate the vertical spreads

$$VS_{i,j} = \frac{C_{i-1,j} - C_{i,j}}{K_i - K_{i-1}},$$

butterfly spreads

$$BS_{i,j} = C_{i-1,j} - \frac{K_{i+1} - K_{i-1}}{K_{i+1} - K_i} C_{i,j} + \frac{K_i - K_{i-1}}{K_{i+1} - K_i} C_{i+1,j},$$

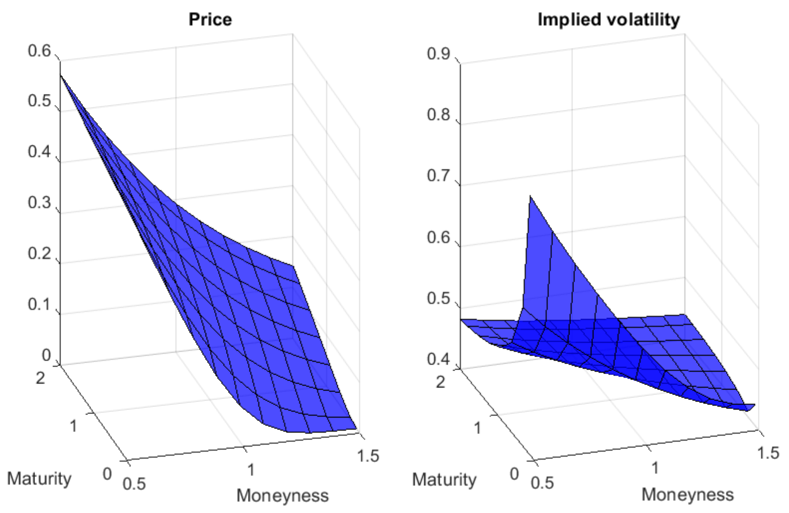


Figure 3.3: Price and implied volatilities for different strikes and maturities

and calendar spreads

$$CS_{i,j} = C_{i,j+1} - C_{i,j}.$$

If there is no arbitrage, there should be $0 \leq VS_{i,j} \leq 1$ for all $i, j = 0, 1, \ldots$ (it is not satisfied in 838 cases), $BS_{i,j} \geq 0$ for all $i, j = 0, 1, \ldots$ (it is not satisfied in 1 502 cases) and $CS_{i,j} \geq 0$ for all $i, j = 0, 1, \ldots$ (it is not satisfied in 480 cases).

As we mentioned in Section 2.2, for options having different strikes different implied volatility is obtained and this is generally a convex function of the strike price. Thus, it is also verified that the implied volatility is a convex function, which is not satisfied in 2 039 cases. Indexes for which one of the Feller's condition, no-arbitrage condition, or convexity of implied volatility is not satisfied are removed from the training set and saved as **trainset_prices_2.csv** and **trainset_ivols_2.csv**.

Figure 3.4 plots the prices and implied volatility for index 96. Figure 3.4 also shows whether the Feller's condition is satisfied and if implied volatility is a convex function. In the lower half of Figure 3.4 is plotted if there is arbitrage using the vertical, butterfly and calendar spreads (red color indicates arbitrage). For index 96 is not satisfied Feller's condition and convexity of implied volatility and it is found arbitrage using butterfly spreads, for example, for moneyness 0.7 and maturity 0.9.

3.2.2 Scaling

As mentioned before, for numerical purposes, input values are often transformed. Model parameters $\Theta = [v(t), \kappa, \theta, \sigma, \rho]$ are scaled using the formula

$$\text{scale}(\Theta^i) = \frac{2\Theta^i - (\Theta^i_{max} - \Theta^i_{min})}{\Theta^i_{max} - \Theta^i_{min}} \tag{3.2}$$

on the interval $[-1, 1]$, where Θ^i_{max} and Θ^i_{min} are values from Table 3.2. Call option prices $C_{i,j}$ are scaled by the formula

$$\text{scale}(C_{i,j}) = \frac{C_{i,j} - \mathbb{E}[C_j]}{std[C_j]}, \tag{3.3}$$

where i is the maturity index and j is the moneyness index. Scaled parameters of the Heston model and option prices are saved as **trainset_prices_3.csv**.

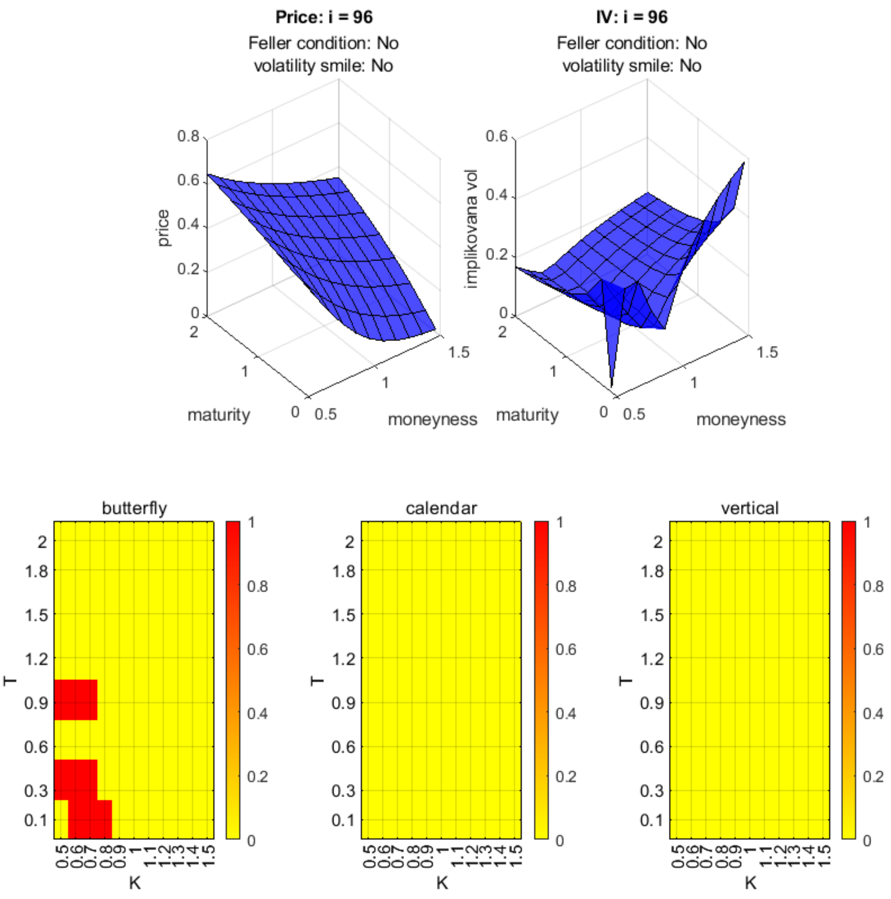


Figure 3.4: Detection of arbitrage, convexity of implied volatility and Feller condition for dataset with index 96

3.3 Neural network architecture

The neural network for option pricing problem has 88 input values (option prices are on a grid 8×11) and 6 output values (five Heston model parameters and an interest rate). For this network, we consider 3 hidden layers, and the number of neurons in each layer is set as an unknown parameter. Experiment Manager is used to train the neural network with the values of the unknown parameters that have been set in Table 3.3. Unknown numbers of neurons in hidden layers are set as multiples of 88 (the number of neurons in the input layer) with decreasing numbers for each subsequent hidden layer. The smallest validation loss (given by the expression (3.1)) is for 176, 44 and 22 neurons in the first, second and third hidden layers.

Table 3.3: Selection of parameter values for neural network training using Experiment Manager

	parameter values
number of neurons in the first hidden layer	$[88, 176, 264, 352]$
number of neurons in the second hidden layer	$[11, 22, 44]$
number of neurons in the third hidden layer	$[11, 22]$

The 5 most commonly used activation functions for neural network training (ReLU, Leaky ReLU, Gaussian error linear unit (GeLU), sigmoid and hyperbolic tangent (tanh)) are considered as activation functions. The comparison of network training for 5 different activation functions is shown in Figure 3.5. The sigmoid function, which has the smallest loss, is hence chosen as the activation function for further experiments.

3.4 Training the network

The input dataset is split into a training (5/6 which is 64 042) and testing (1/6 which is 16 010). Then, a neural network is created with an input layer with 88 neurons, a first, second and third hidden layer of 174, 44 and 22 neurons respectively, and an output layer with 6 neurons. The structure of a network in more detail is described in Section 3.3. As an activation function, the sigmoid function is chosen (see Section 2.4.1 and 3.3) and as an optimizer method, the ADAM method is chosen (see Chapter 2.4.3). Input parameters of the ADAM method (β_1 and β_2), size of minibatch and α, are chosen using the Experiment Manager application in Matlab, which trains the network and calculates the validation loss given by the expression (3.1) for all parameter combinations. The parameter combination with the smallest validation loss is $\beta_1 = 0.8$, $\beta_2 = 0.9$, $minibatch = 50$ and $LR = 0.01$. With these optimal parameters, a neural network is trained with a maximum number of epochs of 10 000. The loss function during the computation is shown in Figure 3.6.

The neural network and the image of the calculation process (loss function) are saved once in 20 epochs. The final trained neural network is saved in **experiment1.mat**.

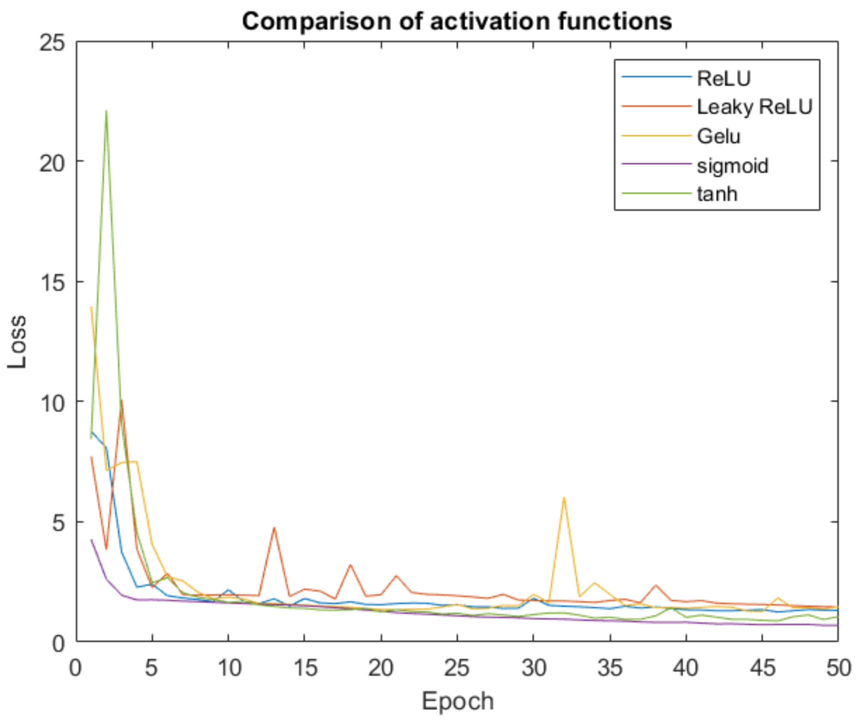


Figure 3.5: Comparison of activation functions for neural network

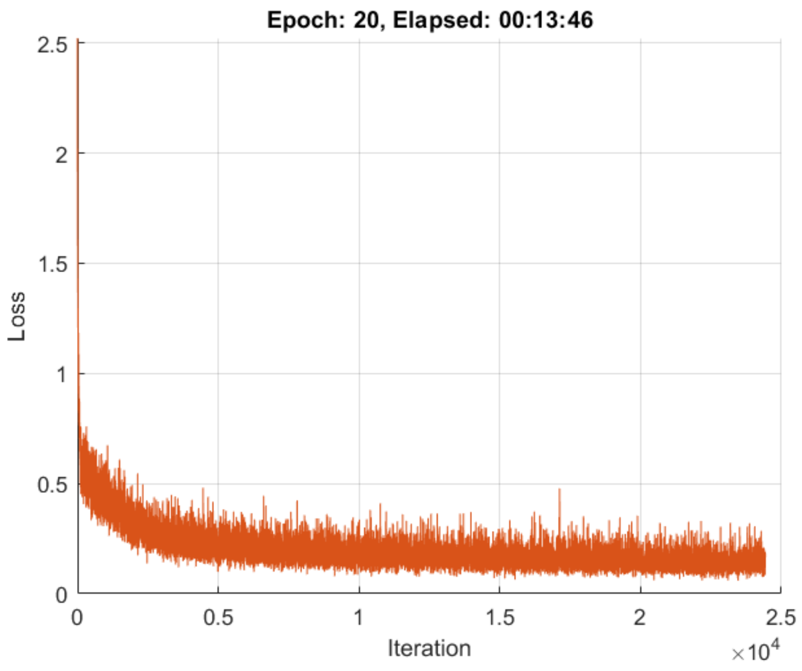


Figure 3.6: Loss function during training of the neural network for option pricing problem

Results 4

This chapter describes the data validation and the graphical representation of the test and predicted values obtained by the trained neural network.

The first section describes the validation of the synthetically generated data, the comparison of the results from the trained network with the test dataset, and the classification of the error. The second section describes the processing of the real market data and their approximation to mapping the 8×11 grid. Then, validation and comparison of the predicted and test real data for the 8×11 grid is performed. Finally, the calibration of the Heston parameters using the *lsqnonlin* function is described. It is determined whether the prices computed from the calibrated parameters match the real input data, and then the option price is computed with these calibrated parameters using a trained neural network.

4.1 Synthetic generated data

In Section 3.2, we described how the synthetic data on which the neural network for option pricing is trained are generated. Data that cause arbitrage are removed, and then both the parameters of the Heston model (including the interest rate) and the option prices are scaled using the formulas (3.2) and (3.3). A neural network architecture is designed with respect to the optimal number of neurons in each hidden layer (see Section 3.3). The NN is then trained as it is described in Section 3.4.

The test data (see data splitting in Section 3.4) is divided into $XTest$ (input option prices) and $YTest$ (Heston model parameters). Using the trained neural network, the parameters of the Heston model, including the interest rate, are predicted from the $XTest$ data and stored in the variable $YPred$. The predicted values correspond to the scaled input parameters of the Heston model, including the interest rate, therefore to estimate the option prices, they need to be scaled back using the formula

$$\Theta^i = \frac{\mathrm{scale}(\Theta^i)(\Theta^i_{max} - \Theta^i_{min})}{2} + \frac{\Theta^i_{max} - \Theta^i_{min}}{2},$$

where Θ^i_{max} and Θ^i_{min} are values from Table 3.2. Now, using the predicted and then back-scaled parameters option prices are calculated using the Heston-Lewis formula (2.6).

We calculate the absolute value of the difference between predicted and input (unscaled) option prices. Figure 4.1 displays the average error (on the left), the standard deviation of the error (in the middle), and the maximum error (on the right) for each option price at the 8×11 grid.

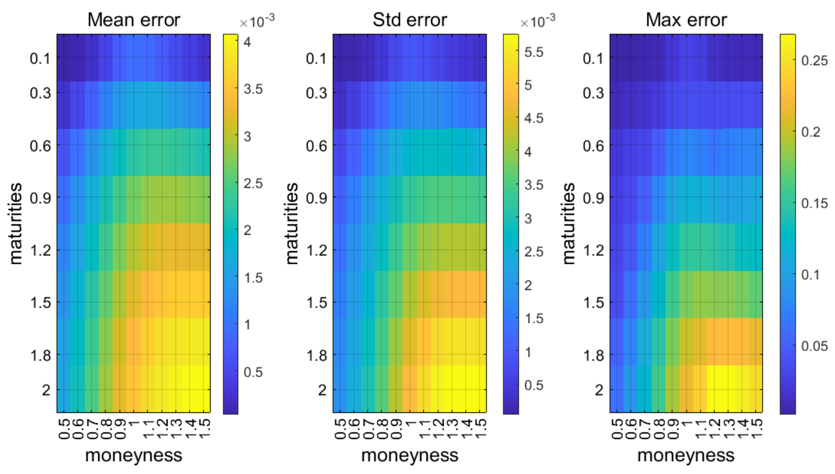


Figure 4.1: Mean, standard deviation, and maximum error for predicted and input option prices

In Figure 4.1, the error increases with increasing time to maturity and moneyness. It can also be noticed that for the mean and standard deviation errors, the error is always less than 6×10^{-3} and the maximum error is less than 0.3 for the whole grid.

4.1.1 Error classification

We define also an overall error for the whole grid as

$$\text{error}_j = \sum_{i=1}^{88} |\text{YTest}_i - \text{YPred}_i|, \tag{4.1}$$

where j is the index in the training set. In the following, we set an error threshold ϵ and count the number of cases (percentage in the training set) for which

$$\text{error}_j \leq \epsilon.$$

The ratio of errors smaller than epsilon is investigated depending on the choice of ϵ. If $\epsilon = 0.88$, which means that each predicted point differs by approximately 0.01, the success rate is 98.71 %. Figure 4.2 shows the sum of errors over the entire grid for each index and the choice of $\epsilon = 0.88$. The biggest sum of errors of all 88 grid values is for index 5 333. The dependence of the success ratio with respect to the choice of epsilon and highlighted value $\epsilon = 0.88$ is depicted in Figure 4.3.

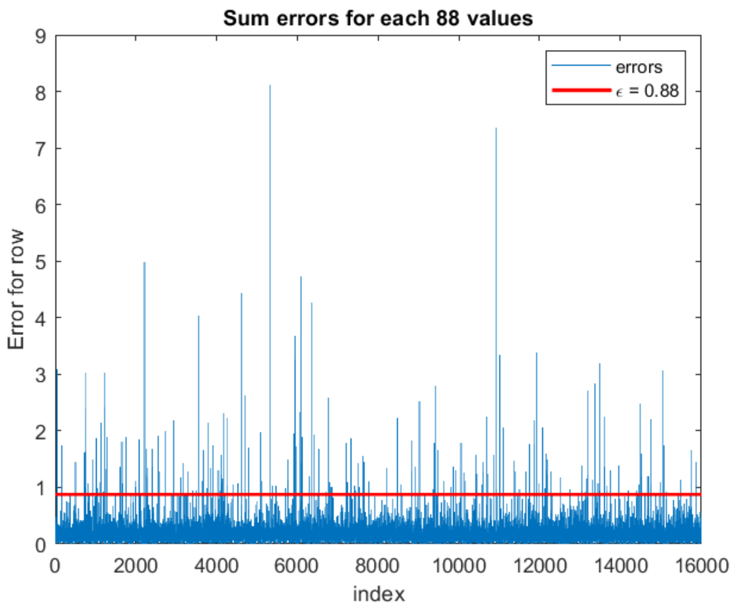


Figure 4.2: Sum of errors over the entire grid (4.1) for each index in the test set with highlighted value $\epsilon = 0.88$

We now compare the input option prices and the prices obtained by the trained neural network. Figure 4.4 shows the price surface cuts for specific maturities (8 maturities, 8 subfigures) for the index with an error around 0.88 (the epsilon threshold) given by expression (4.1). The blue curve describes the input values (the test values from the **trainset_prices_2.csv** file, i.e. the unscaled prices), and the red curve represents the option prices obtained by the trained neural network. The cuts of the price surface for the index 5 333, i.e. index with the maximal error given by expression (4.1), are shown in Figure 4.5. From Figure 4.4 and 4.5, it can be seen that the differences increase with growing maturities, which may be caused by the fact that the longer time to maturity, the greater uncertainty in the prediction.

4.2 Real data

As real data are considered data from the index S&P 500 (Standard & Poor's 500). The index S&P 500 (SPX) is a stock index comprising shares of the 500 largest publicly traded companies in the United States of America.

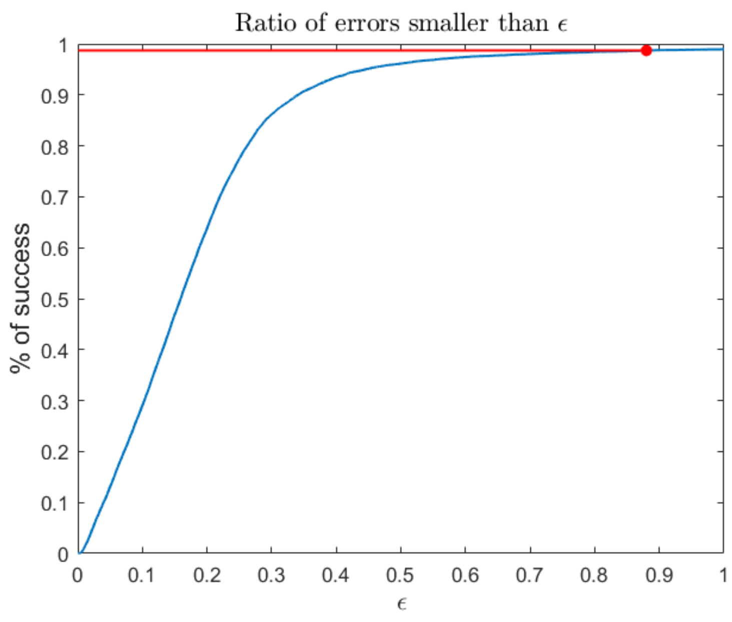


Figure 4.3: Percentage of success with respect to the choice of ϵ with highlighted $\epsilon = 0.88$

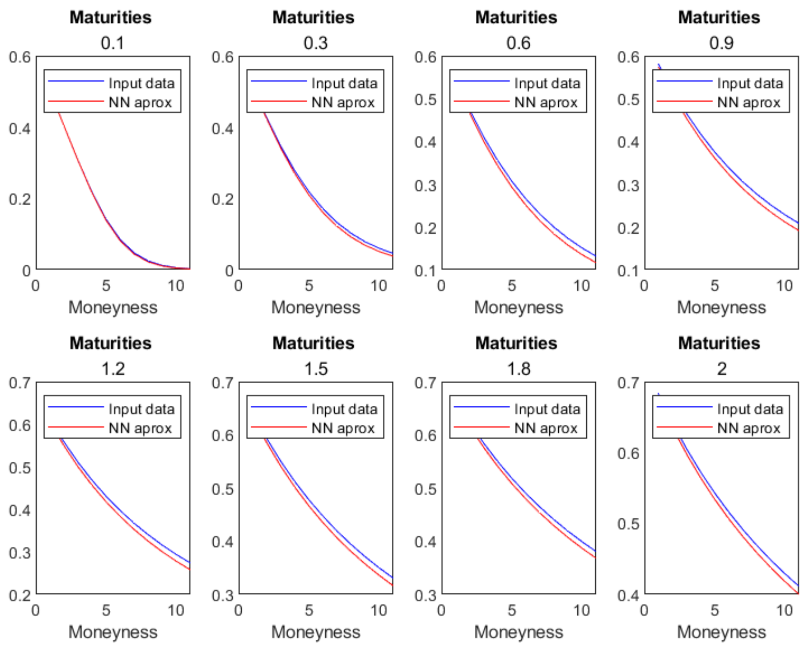


Figure 4.4: Comparing predicted and input values for the index with an error around highlighted value $\epsilon = 0.88$

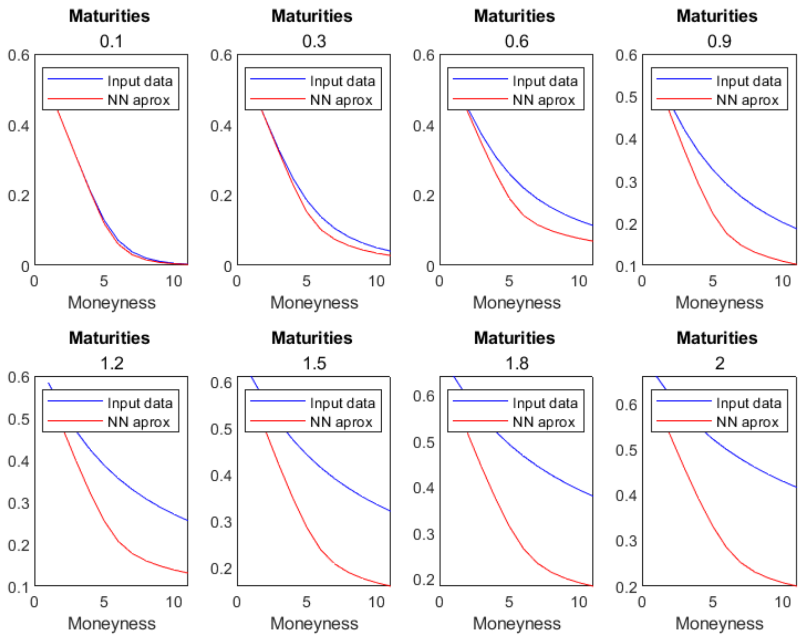


Figure 4.5: Comparing predicted and input values for the index 5 333, i.e. index with the maximal error

4.2.1 Processing of input data

The input data **SPX_S0**[1] is in csv format and contains the columns `trading date`, `open` and `close price` (the first and the last price at which the stock was traded on that day) and `high` and `low price` (the day's highest and lowest trading prices for the stock).

The files **SPX-2019-xx-xx.csv**[2] contains S&P 500 options market data for each trading day of 2019 and are available by the thesis advisor. Each file has a column `date`, `ex-date`, `strike price`, `best bid`, `best offer`, `volume`, `implied volatil-ity`, `forward price` and `time to maturity` (in days).

First, files with empty implied volatility values are detected, and the empty values are replaced by the NaN values (in file **empty_to_NaN.m**). The NaN values of the implied volatility are removed from the data and the time to maturity τ is converted into years using the formula

$$T = \frac{\tau}{252},$$

where 252 is the average number of trading days a year. From formula (2.1), for

[1]`https://www.investing.com/indices/us-spx-500-historical-data` [cit. on 1 May 2023]

[2]`https://optionmetrics.com` [cit. on 18 October 2022]

$t = 0$, we calculate r as

$$r = \frac{1}{T} \ln \left(\frac{FW(0,T)}{S(0)} \right) \tag{4.2}$$

and then we use the Black Scholes formula (2.3) to calculate the price. Moneyness and maturities of the index S&P 500 do not match the values for the generated data, as can be seen in Figure 4.6, where the blue dots show the 8×11 grid for the generated data and the orange dots show the S&P 500 index real market data.

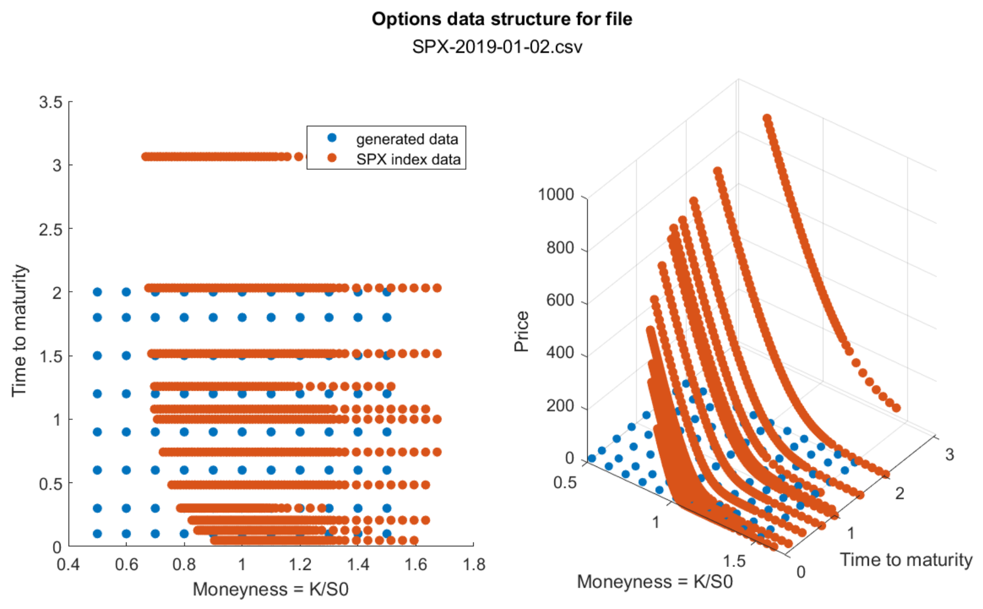


Figure 4.6: Moneyness and maturities for generated data and SPX index

For the forthcoming prediction using the trained neural network, it is necessary to find the option prices of the S&P 500 index for the points on the 8×11 grid. The desired values, denoted as $X[\tau, m]$, are approximated by a linear approximation. First, the closest maturity values are found from real data, i.e. $\tau_i < \tau < \tau_{i+1}$ and from the linear combination

$$\tau = (1 - a)\tau_i + a\tau_{i+1}$$

we calculate

$$a = \frac{\tau - \tau_i}{\tau_{i+1} - \tau_i}.$$

Then the moneyness values, for the closest i-th maturity, are used to determine the closest moneyness from the real data, i.e. $m_j < m < m_{j+1}$ and from the linear combination

$$m = (1 - b)m_j + bm_{j+1}$$

we calculate

$$b = \frac{m - m_j}{m_{j+1} - m_j}.$$

And the same is done for $i + 1$-th maturity. From the moneyness values are determined the closest moneyness values from real data, i.e. $m_k < m < m_{k+1}$ and from a linear combination

$$m = (1 - c)m_k + cm_{k+1}$$

are calculated

$$c = \frac{m - m_k}{m_{k+1} - m_k}.$$

The wanted points $X[\tau, m]$ lie inside a quadrilateral with vertices $A[\tau_i, m_j]$, $B[\tau_i, m_{j+1}]$, $D[\tau_{i+1}, m_k]$, $C[\tau_{i+1}, m_{k+1}]$, see Figure 4.7.

Finally, a line is constructed perpendicular to the parallels of the maturity lines, so that it passes through the wanted point $X[\tau, m]$. The intersection of the perpendicular line and the line τ_i is denoted as P and the intersection of the perpendicular line and the line τ_{i+1} is denoted as Q. Using a, b, c calculated above, we calculate

$$\text{Price\_P} = (1 - b)\text{Price\_A} + b\text{Price\_B},$$

$$\text{Price\_Q} = (1 - c)\text{Price\_D} + c\text{Price\_C},$$

and finally

$$\text{Price\_X} = (1 - a)\text{Price\_P} + a\text{Price\_Q}.$$

The calculation process is saved in file **grid_price.m** and depicted in Figure 4.7.

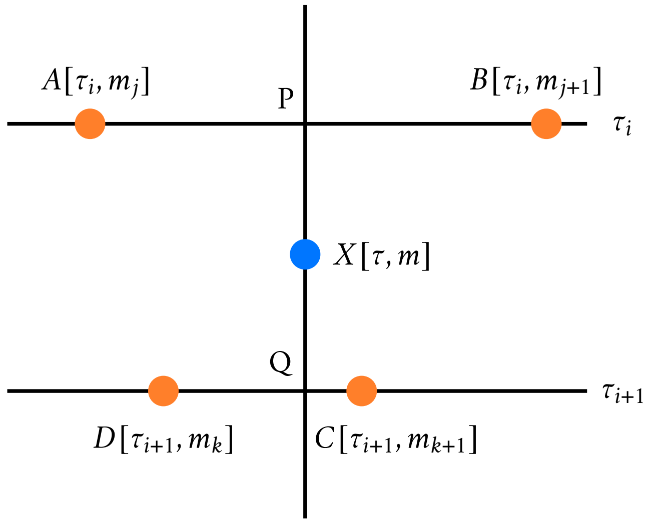


Figure 4.7: Linear approximation at the point $X[\tau, m]$

The option prices of index S&P 500 converted to a grid of generated data are plotted in Figure 4.8. For the grid of 8 maturities and 11 moneyness, are not all values calculated because some wanted points do not have all neighboring points $A[\tau_i, m_j]$, $B[\tau_i, m_{j+1}]$, $C[\tau_{i+1}, m_{k+1}]$ and $D[\tau_{i+1}, m_k]$ available.

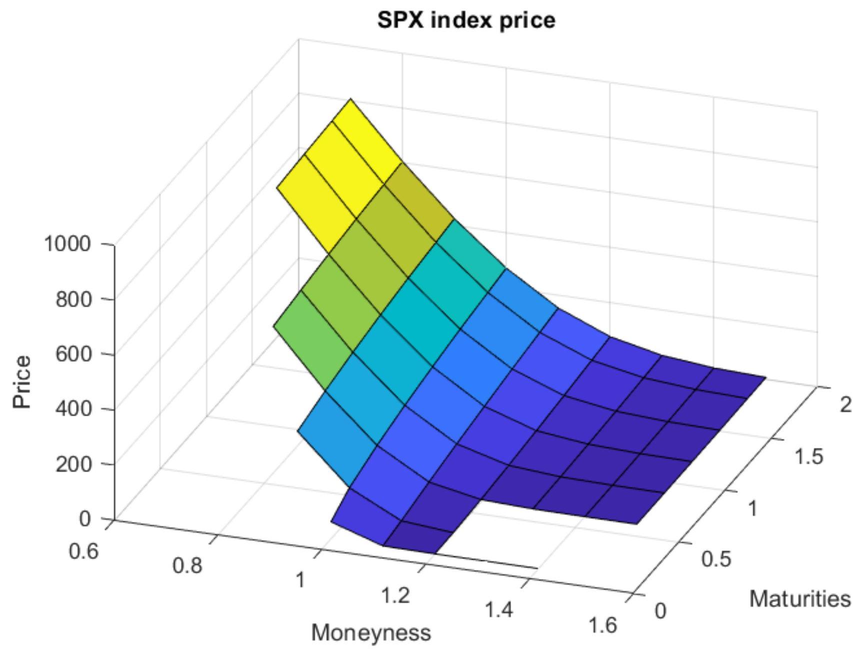


Figure 4.8: SPX index price converted to a grid of generated data

The missing SPX index option prices on the 8 × 11 grid are calculated for each moneyness using a system of equations

$$\begin{aligned} am_1 + b &= p_1 \\ am_2 + b &= p_2 \end{aligned} \tag{4.3}$$

where m_1 and m_2 are the last and penultimate moneyness with a known option price value p_1 and p_2. First, we express

$$a = \frac{p_1 - b}{m_1}$$
$$b = \frac{p_2 m_1 - p_1 m_2}{m_1 - m_2}$$

from the system of equations (4.3) and then use the constants a and b to calculate the missing option price values on the 8 × 11 grid. In some cases, using a system of equations (4.3) we get negative values, which is not appropriate for price approximation. In these cases, prices are defined linearly using the `linspace` command from the last known value to 0.001. The option prices of the SPX index on the 8×11 grid are saved in the variable `PRICE`. For indexes 94, 157 and 221, there is no value for maturity 0.1, so the values cannot be approximated using this procedure, and the indexes are removed from validation due to missing real data.

It is worth to mention that any linear interpolation and especially extrapolation of this type is highly unreliable due to the nonlinear character of the price surfaces and it is presented here for demonstration purposes. Testing higher order interpolation and extrapolation methods was beyond the scope of the thesis.

For synthetically generated data, the stock price at time 0 is assumed to be equal to 1, but this is not satisfied for real market data, where we have $S(0)$ different for each day. From formula (2.6) it follows that

$$C(0, S(0), K) = S(0)C\left(0, 1, \frac{K}{S(0)}\right)$$

where K can be rewritten as $m \cdot S(0)$

$$C(0, S(0), m \cdot S(0)) = S(0)C(0, 1, m) \tag{4.4}$$

and after dividing by $S(0)$, we obtain

$$C(0, 1, m) = C(0, S(0), m \cdot S(0))/S(0). \tag{4.5}$$

Thus, to get the option prices calculated the same way as the generated data, we need to divide the prices in the variable `PRICE` by $S(0)$. The prices from the variable `PRICE` are stored after dividing by $S(0)$ in the file **PRICE.mat**.

4.2.2 Validation

First, **experiment1.mat** containing the neural network trained with synthetic data (see Section 3.4) and **PRICE.mat** that contains option prices approximated on an 8×11 grid after recalculation using formula (4.5) are loaded. Then the prices are scaled using the formula (3.3) and the loaded **training_mean.mat** and **training_std.mat** files (mean and standard deviation used in scaling the training synthetic data). From these prices, 6 parameters are estimated using a trained neural network. The parameters r, $v(0)$, κ, θ and σ are back-scaled (the parameter ρ does not need to be scaled, it is generated on the interval $[-1, 1]$). Using the back-scaled parameters, the option price is calculated by the Heston-Lewis formula (2.6) and saved in **validation_real_data.mat**.

The prices stored in **PRICE.mat** and **validation_real_data.mat** are recalculated back to prices of the index S&P 500 according to formula (4.4), i.e. multiplied by $S(0)$. For these prices, the error is calculated as the difference in absolute value. The maximum error is 5×10^{12}, which is caused by index 241 (18 December 2019, the fourth tripple-witching[3] day in 2019). For index 241 with maturities equal to 0.1 and 0.3, the neural network returns parameters from which the calculated prices are negative and do not match the input values, as shown in Figure 4.9, where the blue curves show the option prices of the index S&P 500 and the red curves show the option prices obtained by the neural network.

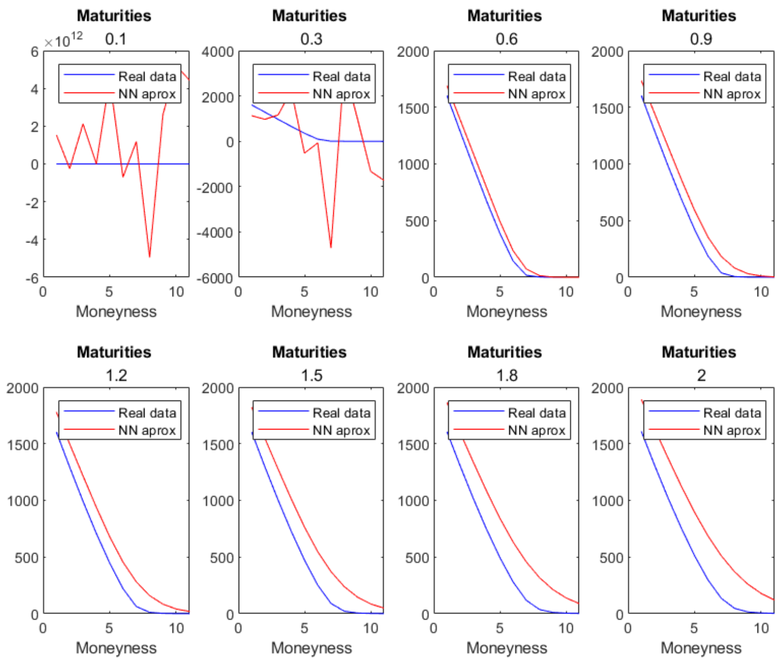


Figure 4.9: Comparing predicted and input values for the index 241 with the maximal error

The parameters estimated by the neural network for index 241 ($r = 1.04$, $v(0) = -1.31$, $\kappa = -0.55$, $\theta = -0.91$, $\sigma = -1.11$ and $\rho = 0.11$) are very different than the estimated parameters for nearby indexes (days), even though the input prices to the neural network of nearby indexes (days) are very similar. Thus, the error is in the estimation of the parameters by the trained neural network, not in the subsequent calculation of prices from the estimated parameters.

If the index 241 is removed, we get a maximum error equal to 1 358.9. The average error, standard deviation, and maximum error for the entire 8×11 grid after removing index 241 are shown in Figure 4.10. From Figure 4.10, it can be seen that the mean error and the standard deviation error are less than 550 for each grid point. The price surface cuts for specific maturities are depicted for the index 189 (index with the error closest to the mean error over the whole grid) in Figure 4.11,

[3]`https://www.investopedia.com/terms/t/triplewitchinghour.asp` [cit. on 8 May 2023]

where the blue curves show the option prices of the index S&P 500 and the red curves show the option prices obtained by the neural network.

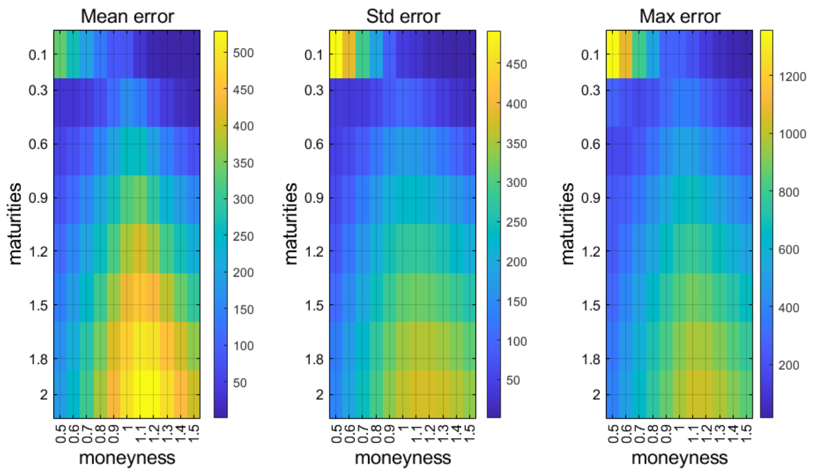


Figure 4.10: Mean, standard deviation, and maximum error for predicted and real input option prices

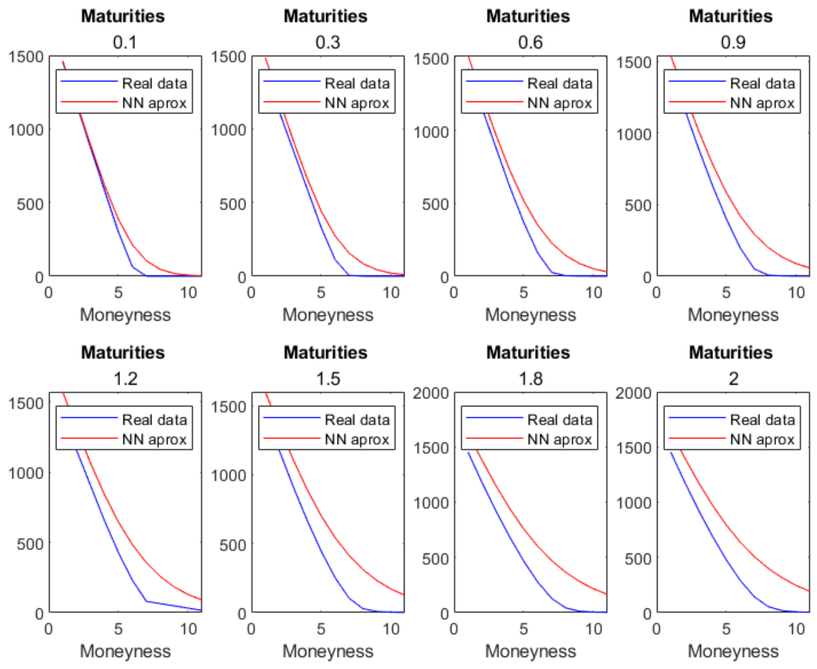


Figure 4.11: Comparing predicted and input values for the index 189, i.e. index with error closest to the mean error

4.2.3 Calibration without neural networks

Since the prices obtained by the neural network are inaccurate when applied to real market data, the process of estimating the 6 parameters by the neural network is replaced by the classical calibration of the Heston model as it is described for example in [22]. In the file `Real data\Calibration\Data preparation\editing_data.m` the input data for calibration are modified, i.e. the columns contain `maturity` (in years), `strike price`, `best bid`, `best offer`, `r` (calculated using the formula (4.2)), `price` (calculated using the Black Scholes formula (2.3)) and `S(0)` (spot price at time 0). The edited data for calibration are stored for each trading day of 2019 in the `Data - calibration` folder.

The calibration process is described in the file `Real data\Calibration\Run Calibration`. The calibration is run in the file `run.m`, which calls the function `cal.m` with the parameters `dataid` – id of input data (SPX), `from` – start date, `to` – end date, `model` – model type (Heston), `weights` – input parameter for the *hestonUtility* function, `resultsfname` – the filename in which the results are stored and `runs` – number of calibration for each day. In the function `call.m`, the data for the calibration is first loaded, followed by defining the ranges of the 5 Heston parameters and the output files. Then an initial condition x_0 is chosen for the function *lsqnonlin* (nonlinear least squares), which finds the parameters so that the differences between the input data and the data calculated using the Heston-Lewis formula are minimized. These optimal parameters are then stored in the output file **result-SPX-Heston.csv**. The calibration process is described in more detail in the paper [22].

The real data are compared with the prices calculated from the calibrated parameters, i.e. it is determined how well the parameters are estimated by the calibration. The calibrated parameters, the real data, and the data containing the spot prices $S(0)$ are loaded. For the real data, the rows containing NaN values for the implied volatility are removed and sorted by maturity and then by the strike price. From these data, r is calculated using the formula (4.2) and the price of real data using the Black Scholes formula (2.3). Finally, the price is calculated from the calibrated parameters using the Heston-Lewis formula (2.6). A comparison of the price calculated from the real data and the price calculated using the calibrated parameters is depicted in Figure 4.12. From Figure 4.12, it can be seen that the prices are almost identical to each other, which is true for all trading days of 2019 (see file `calibration_and_real_data.m`).

Since the parameters of the Heston model are well-calibrated, we will compare the prices calculated from the calibrated parameters with the prices found by the neural network. First, r is defined as the average of the r's computed from the real data using the formula (4.2). Using the calculated r and calibrated parameters, the price on the 8×11 grid is calculated by the Heston-Lewis formula (2.6) and stored

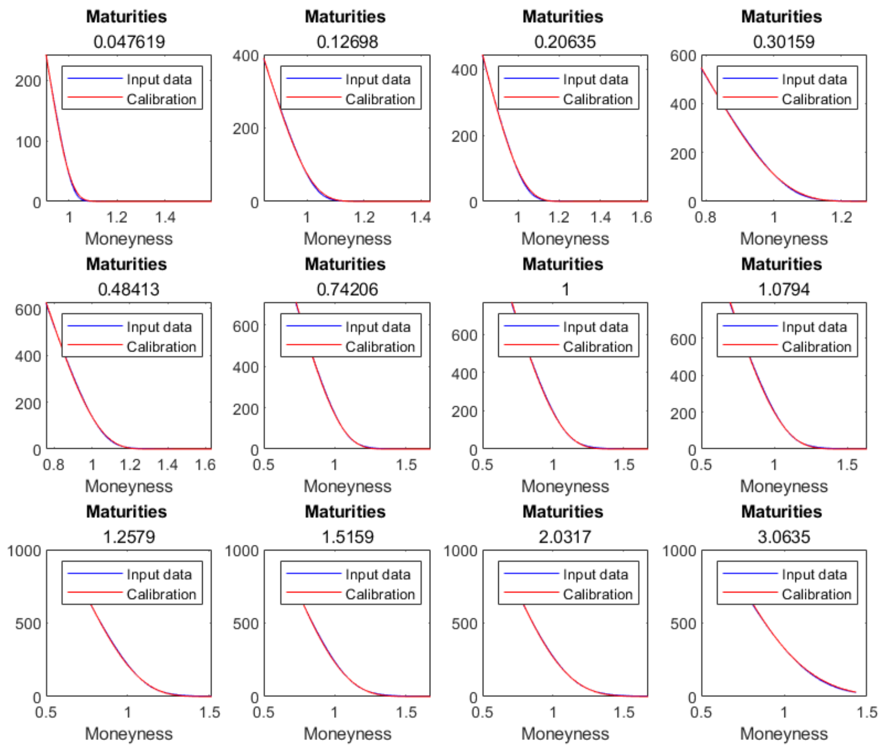


Figure 4.12: Comparison of the price calculated from real data and the price calculated using calibrated parameters for 2 January 2019

in the variable `calibration_price`.

Now, the prices are obtained using the trained neural network, i.e. the values in variable `calibration_price` are scaled using the files **training_mean.mat** and **training_std. mat** (the mean and standard deviation used in scaling the training synthetic data), from these prices the parameters are estimated using the trained neural network, these parameters are scaled back using the values in the Table 3.2, and the option price is estimated from these parameters using the Heston-Lewis formula (2.6). The prices obtained by the neural network and calibration are stored in file **NN_price_all.mat** and **calibration_price_all.mat**. The error is defined as the difference between these prices in absolute value. The average error, standard deviation, and maximum error for the entire 8×11 grid is depicted in Figure 4.13. The maximum error for the entire grid was reduced from 1 358.9 to 149.29 compared to the validation in Section 4.2.2. The price surface cuts for maturities are depicted for the index 150 (index with the error closest to the mean error over the whole grid) in Figure 4.14.

For Figures 4.13 and 4.14, it can be seen that the error is significantly lower

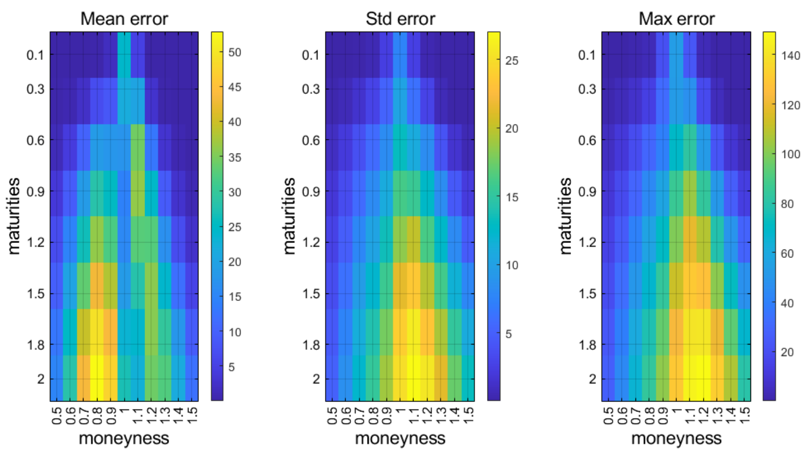


Figure 4.13: Mean, standard deviation and maximum error for prices predicted using neural network and prices calculated from calibrated parameters

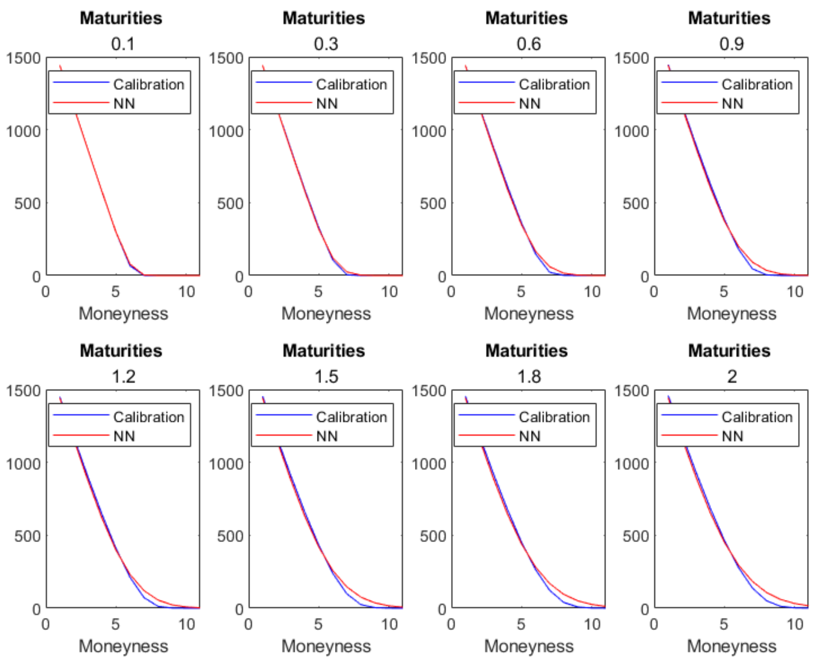


Figure 4.14: Comparing prices predicted using neural network and prices calculated from calibrated parameters for the index 150, i.e. index with error closest to the mean error

than in Figures 4.9 and 4.10, i.e. the inaccurately estimated option prices in the Section 4.2.2 are due to the linear approximation of the real data to an 8×11 grid.

Conclusion 5

In the presented Master's thesis, we studied option pricing in the stochastic volatility Heston model using neural networks. In Chapter 2, the basic definitions from stochastic calculus were presented together with the introductions of the Black-Scholes and the Heston models. Subsequently, the fundamental neural networks principals and techniques were explained.

Chapter 3 describes how to select the optimal parameters for training the neural network with the help of the Experiment Manager. An illustrative example is presented: approximation of a smooth nonlinear function of two variables using a neural network. Using the optimal combination of parameters, the neural network is trained, and a prediction is made on the test data. The minimum, mean and maximum values of the difference between the predicted and tested values for the function are 4.54×10^{-9}, 2.43×10^{-5} and 1.99×10^{-4} respectively. In the second part of the chapter, we looked at option pricing. First, synthetic option prices in Heston model were generated using the Heston-Lewis formula for different combinations of parameters. Then, the data were removed for which the Feller condition, the no-arbitrage condition, or the convexity of implied volatility is not satisfied, and the prices and parameters of the Heston model were scaled. Finally, the optimal parameter values for training the option prices were found using the Experiment Manager application (number of neurons in the first/second/third hidden layer 176/44/22, $\beta_1 = 0.8$, $\beta_2 = 0.9$, $minibatch = 50$ and $LR = 0.01$) and the neural network was trained with these parameters.

Chapter 4 describes the data validation and graphical representation of the test and predicted values obtained by the trained neural network for synthetically generated data and then for real market data (index S&P 500). In the first part of the chapter, we consider the validation of synthetically generated data, i.e. using a neural network, option prices are predicted and compared to the test data. The maximum error, defined as the difference in absolute value between the predicted and test values, is less than 0.3 and the average error is less than 4×10^{-3}. The error over the entire grid defined by expression (4.1) is less than 0.88, which means that each predicted point differs by approximately 0.01, in 98.71 % of the cases.

In the second part of Chapter 4, we present results for real market data of the S&P 500 index. Since the maturities and moneyness pairs in the real market data do not match the maturities and moneyness pairs of the grid used in the trained neural network input layer, a linear approximation of the real option prices to the 8×11 grid was performed. Using a neural network, we estimated the parameters from option prices of the index S&P 500 on the 8×11 grid and used the Heston-Lewis formula to back-calculate the prices from these estimated parameters. We showed that the linear interpolation and especially the extrapolation of points on a generally highly nonlinear price surface lead to huge errors of the neural network approximation.

In the third part of Chapter 4, the classical calibration of Heston parameters using the function *lsqnonlin* (nonlinear least squares, i.e. without neural networks) was described. The prices calculated from the calibrated parameters are very close to the real market data. Since the parameters of the Heston model are well-calibrated, we used them to calculate the prices directly at the grid points used in the trained neural network input layer. These prices calculated from the calibrated parameters and the prices obtained using the neural network approximation were compared. The maximum error for the entire grid was reduced from 1 358.9 to 149.29 compared to the validation in Section 4.2.2, i.e. the inaccurately estimated prices from Section 4.2.2 are not caused by the neural network but by the linear approximation of the real data to the 8×11 grid.

For a more accurate approximation, for example, a higher-order polynomials or splines could be used. Currently, no reliable algorithm is available for approximating the option price surface, and the task of finding an optimal algorithm is beyond the scope of this thesis. Further issues also include converting prices into implied volatilities and training the neural network directly with implied volatilities. However, there is no exact formula for calculating implied volatilities, and they have to be solved numerically, which implies further additional errors. Another improvement would be to consider different interest rates for different maturities, since real market data usually have this property. However, it is worth to mention, that presented methodology can be easily modified to re-price the prices surface at each maturity , although the numerically unreliable calculation of implied volatilites would have to be performed in this task.

Thesis attachment A

The attached DVD contains the following files:

- Bacova_DP_2023.pdf – this Master's thesis
- Readme.txt – information about each attached file in a tree structure

and the following folders:

- Data
- Data – arbitrage
- Data – calibration
- Figures
- Function
- Real data
- Synthetic generated data

The `Data` folder contains the spot prices $S(0)$ of the SPX index for each trading day of 2019 (**SPX_S0.csv**), input data for the hypotenuse function from Section 2.4.4 (**trainset_hypotenuse.csv**), and generated data for option prices and implied volatilities (**trainset_prices.csv** and **trainset_ivols.csv**), including edited data after removing the arbitrage and scaling (**trainset_prices_2.csv**, **trainset_ivols_2.csv** and **trainset_prices_3.csv**). All option price input data **SPX_2019-xx-xx.csv** are available by the thesis advisor on request.

In the folder `Data -- arbitrage` there are the indexes and arrays of moneyness and maturity for which no-arbitrage conditions are not satisfied. The `Data -- calibration` folder contains the modified input data for calibration. In the folder `Figures` are codes for generating some Figures (more information in `Readme.txt`). The `Function`, `Synthetic generated data` and `Real data` folders contain all the necessary *.m* and *.mat* files for data generation, data editing, running neural network training, validation and for graphical outputs.

By typing **help function_name.m** in the Matlab console, you can get documentation for all Matlab functions used in this thesis.

Experiment Manager B

To run the Experiment Manager Application in Matlab, Deep Learning Toolbox[1] is needed. It can be found at the `APPS` tab under the `Experiment manager` button, see Figure B.1.

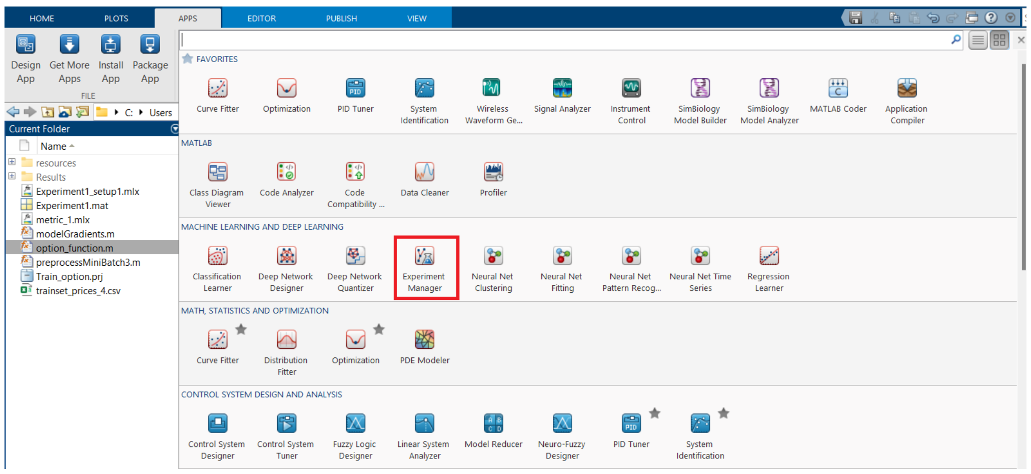


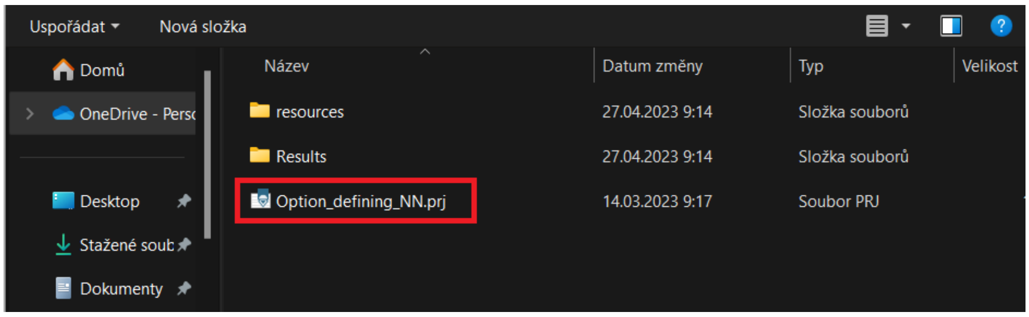
Figure B.1: Instructions to start the Experiment Manager App – part 1

To open a project click on `Open...` and select the file with the extension *.prj*.

In the `Experiment1` tab, in the Hyperparameters section, one can choose the values of unknown parameters (in this case the number of neurons in the first, second and third hidden layer) and in the Setup Function section is a *.m* function that is run with all combinations of the selected hyperparameters.

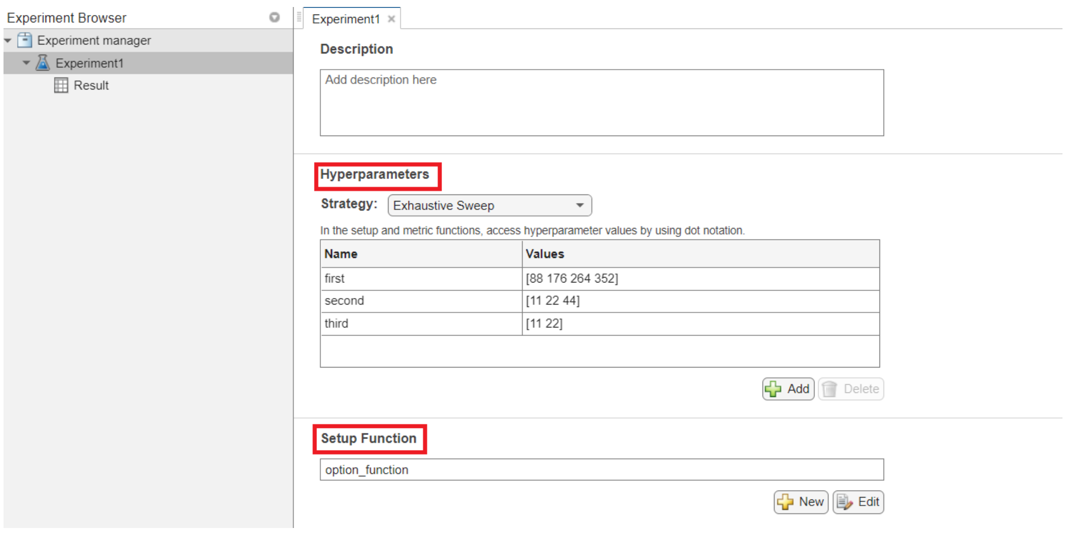
The `Result` tab shows the neural network training results for all combinations of the selected hyperparameters. Training/validation RMSE and loss are displayed in the right part of the table and the option to view the progress of the validation RMSE and loss during training is in the `Training plot` tab.

[1]see `https://www.math-works.com/products/deep-learning.html`



Název	Datum změny	Typ	Velikost
resources	27.04.2023 9:14	Složka souborů	
Results	27.04.2023 9:14	Složka souborů	
Option_defining_NN.prj	14.03.2023 9:17	Soubor PRJ	

Figure B.2: Instructions to start the Experiment Manager App – part 2



Description

Add description here

Hyperparameters

Strategy: Exhaustive Sweep

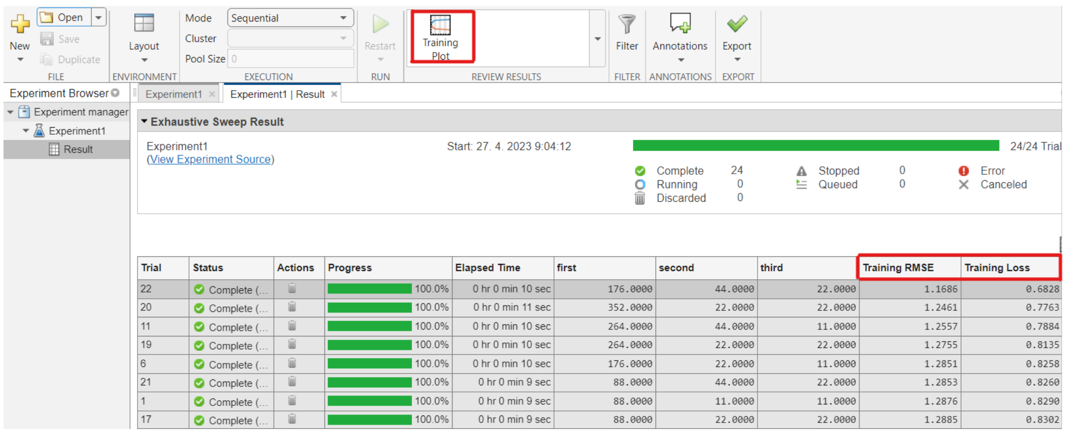
In the setup and metric functions, access hyperparameter values by using dot notation.

Name	Values
first	[88 176 264 352]
second	[11 22 44]
third	[11 22]

Setup Function

option_function

Figure B.3: Instructions to start the Experiment Manager App – part 3



Exhaustive Sweep Result

Experiment1
(View Experiment Source)

Start: 27. 4. 2023 9:04:12

24/24 Trial

Complete	24	Stopped	0	Error	
Running	0	Queued	0	Canceled	
Discarded	0				

Trial	Status	Actions	Progress	Elapsed Time	first	second	third	Training RMSE	Training Loss
22	Complete (...		100.0%	0 hr 0 min 10 sec	176.0000	44.0000	22.0000	1.1686	0.6828
20	Complete (...		100.0%	0 hr 0 min 11 sec	352.0000	22.0000	22.0000	1.2461	0.7763
11	Complete (...		100.0%	0 hr 0 min 10 sec	264.0000	44.0000	11.0000	1.2557	0.7884
19	Complete (...		100.0%	0 hr 0 min 10 sec	264.0000	22.0000	22.0000	1.2755	0.8135
6	Complete (...		100.0%	0 hr 0 min 10 sec	176.0000	22.0000	11.0000	1.2851	0.8258
21	Complete (...		100.0%	0 hr 0 min 9 sec	88.0000	44.0000	22.0000	1.2853	0.8260
1	Complete (...		100.0%	0 hr 0 min 9 sec	88.0000	11.0000	11.0000	1.2876	0.8290
17	Complete (...		100.0%	0 hr 0 min 9 sec	88.0000	22.0000	22.0000	1.2885	0.8302

Figure B.4: Instructions to start the Experiment Manager App – part 4

Bibliography

1. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. New York: Springer-Verlag, 2004. Springer Finance. ISBN 978-0-387-40101-0. ISSN 1616-0533.
2. HEBB, D.O. *The Organization of Behavior*. New York: Wiley, 1949. ISBN 978-1-135-63190-1.
3. ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*. 1958, vol. 65, no. 6, pp. 386–408. ISSN 0033-295X. Available also from: `http://dx.doi.org/10.1037/h0042519`.
4. ROSENBLATT, F. Principles of statistical neurodynamics. *Spartan Books*. 1962, vol. 65, no. 6, pp. 386–408. ISSN 0033-295X. Available also from: `http://catalog.hathitrust.org/Record/000203591`.
5. RUMELHART, David E.; HINTON, Geoff E.; WILSON, R. J. Learning representations by back-propagating errors. *Nature*. 1986, vol. 323, pp. 533–536.
6. CYBENKO, G. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*. 1989, vol. 2, pp. 303–314.
7. HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks*. 1989, vol. 2, no. 5, pp. 359–366.
8. LU, Zhou; PU, Hongming; WANG, Feicheng; HU, Zhiqiang; WANG, Liwei. The Expressive Power of Neural Networks: A View from the Width. In: GUYON, I. et al. (eds.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, vol. 30. Available also from: `https://proceedings.neurips.cc/paper/2017/file/32cbf687880eb1674a07bf717761dd3a-Paper.pdf`.
9. BLACK, Fischer; SCHOLES, Myron. The pricing of options and corporate liabilities. *J. Polit. Econ.* 1973, vol. 81, no. 3, pp. 637–654. ISSN 0022-3808.
10. HULL, John Campbell; WHITE, Alan D. The pricing of options on assets with stochastic volatilities. *J. Finance*. 1987, vol. 42, no. 2, pp. 281–300. ISSN 1540-6261.

11. CHESNEY, Marc; SCOTT, Louis. Pricing European currency options: A comparison of the modified Black-Scholes model and a random variance model. *J. Financ. Quant. Anal.* 1989, vol. 24, no. 3, pp. 267–284. ISSN 0022-1090.
12. HESTON, Steven L. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Rev. Financ. Stud.* 1993, vol. 6, no. 2, pp. 327–343. ISSN 0893-9454.
13. RUF, Johannes; WANG, Weiguan. Neural networks for option pricing and hedging: a literature review. *The Journal of Computational Finance*. 2020.
14. HORVATH, Blanka; MUGURUZA, Aitor; TOMAS, Mehdi. Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models. *Quant. Finance*. 2021, vol. 21, no. 1, pp. 11–27. ISSN 1469-7688. Available from DOI: `10.1080/14697688.2020.1817974`.
15. DESPRÉS, Bruno. *Neural Networks and Numerical Analysis*. Vol. 6. Berlin: De Gruyter, 2022. De Gruyter Ser. Appl. Numer. Math. ISBN 978-3-11-078312-4. ISSN 2512-1820.
16. LEWIS, Alan L. *Option Valuation Under Stochastic Volatility: With Mathematica code*. Newport Beach, CA: Finance Press, 2000. ISBN 9780967637204.
17. SHREVE, Steven E. *Stochastic calculus for finance II. Continuous-time models.* New York: Springer-Verlag, 2004. Springer Finance. ISBN 978-1-441-92311-0. ISSN 1616-0533.
18. RUDER, Sebastian. *An overview of gradient descent optimization algorithms*. 2016. Available at arXiv: `https://arxiv.org/abs/1609.04747`.
19. MERTON, Robert C. Theory of rational option pricing. *Bell J. Econ.* 1973, vol. 4, no. 1, pp. 141–183. ISSN 0005-8556.
20. COX, J. C.; INGERSOLL, J. E.; ROSS, S. A. A theory of the term structure of interest rates. *Econometrica*. 1985, vol. 53, no. 2, pp. 385–407. ISSN 0012-9682. Available from DOI: `10.2307/1911242`.
21. CS231N. CS231n Convolutional Neural Networks for Visual Recognition Course Website. 2022. Available also from: `https://cs231n.github.io/neural-networks-3/`.
22. MRÁZEK, Milan; POSPÍŠIL, Jan. Calibration and Simulation of Heston Model. *Open Math.* 2017, vol. 15, no. 1, pp. 679–704. ISSN 2391-5455. Available also from: `https://www.degruyter.com/view/j/math.2017.15.issue-1/math-2017-0058/math-2017-0058.xml`.

List of Figures

List of Tables