FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Applications of Deep Learning in Option Pricing and Calibration

Tobias Cramer Pedersen & Magnus Grønnegaard Frandsen

Project Outside Course Scope, Mathematics-Economics
Department of Mathematics, University of Copenhagen
Supervisor: Rolf Poulsen

5th of April, 2020

Applications of Deep Learning in Option Pricing and Calibration

Tobias Cramer Pedersen & Magnus Grønnegaard Frandsen
Department of Mathematical Sciences, University of Copenhagen

5th of April, 2020

Abstract

In this project, artificial neural networks (ANN) are applied in the setting of option pricing and model calibration.

In option pricing, the aim is to utilize ANNs as a means to calculate prices and their corresponding deltas. The use of deep and differential ANNs yields a significant improvement in accuracy over shallow ANNs and enables accurate estimation of greeks. Without variance reduction, the accuracy of ANNs is inferior to ordinary Monte Carlo simulations, especially for deep out-of-the-money call options, but this is remedied with the utilization of Sobol numbers and averaging ANNs. It is also shown that accuracy largely dependent on the sample data and training procedures.

In model calibration, the aim is to construct an ANN-based calibration method that performs robust and fast calibrations to a volatility surface. The performance of the calibration method is analyzed in relation to a benchmark procedure. The method improves the computational speed by orders of magnitude but is not as precise. The findings suggest that ANNs should be considered as a potential replacement for the benchmark procedure.

***Keywords*—** Artificial Neural Network, Backpropagation, Option Pricing, Differential Neural Networks, Optimal Depth of Neural Networks, Model Calibration, Heston Model.

Contents

1 Introduction

The applicability of artificial neural networks (ANN) in areas of finance is suggested by various authors. As one deviates from the simple Black and Scholes framework or considers complex derivatives, the otherwise nicely behaved closed-form solutions are replaced by unstable numerical integrals or slow Monte Carlo methods. As the computational speed of a forward pass (evaluation of an ANN) is relatively fast compared to the often computationally heavy calculations in finance, the idea of replacing the traditional methods with artificial neural networks is appealing.

Even though artificial neural networks are, in theory, capable of approximating any reasonably behaved function, it is much more difficult in practice to accurately capture and learn the behavior of multidimensional functions and even one-dimensional derivatives. This project aims to explore the challenges of utilizing artificial neural networks in finance from a theoretical and practical perspective.

In section 2, the theoretical framework for working with artificial neural networks in regression is presented, including a thorough derivation of the backpropagation algorithm. This section also presents the complexities behind the comparison between deep and shallow artificial neural networks from a theoretical perspective [7].

In section 3, the possibility of having an artificial neural network learn the option pricing function from Monte Carlo simulated option payoffs is explored. The framework is used to test the relationship between depth and width from a financial perspective. In [3], it is suggested that utilizing derivatives of payoffs, through so-called differential ANNs, might have a strong stabilizing and regularizing effect, which might improve accuracy. In section 3, we also seek to justify the theoretical use of differential ANNs and compare their accuracy to ordinary feedforward ANNs for option pricing.

In section 4, the computationally heavy procedure of calibrating a stochastic volatility model to a volatility surface is considered like in [1] and [2]. An analysis of whether or not the artificial neural networks can learn the complex behavior of the volatility surface of the Heston model as a function of the model parameters sufficiently enough to calibrate the model is conducted.

All ANN implementations in this article are implemented in Python 3.6 utilizing Tensorflow 1.15.0 and Keras 2.3.1. The implementations are provided on Github in the following repo:

`https://github.com/tcpedersen/ANNOptionPricingAndCalibration`

Both authors are involved in and responsible for all aspects of the project. The code and formulations of section 3 are mostly produced by Magnus Grønnegaard Frandsen. The code and formulations of section 4 are mostly produced by Tobias Cramer Pedersen.

2 Artificial Neural Networks

This section will focus on the theoretical issues of utilizing artificial neural networks for regression. It will also introduce the notation used throughout the project. The section is divided into six parts

1. Introduction of the multidimensional regression problem.
2. Introduction of feedforward ANN as a parametric family of functions.
3. A presentation of a sufficiently strong Universal Representation Theorem, which ensures that the ANNs considered in this article are capable of approximating any continuous function to arbitrary precision.
4. A detailed derivation and discussion of a regression/classification problem, which is best solved by a deep ANN. The section is meant to strengthen our understanding of the complexities involved in the comparison between shallow and deep neural networks.
5. A thorough derivation of the backpropagation algorithm used when training ANNs.
6. A summary of the practical training procedures utilized in this project.

2.1 The Multidimensional Regression Problem

In a general regression setting, we consider possessing m samples of n_0 dimensional features organised in a matrix $X = [x^{(1)}, \dots, x^{(m)}] \in \mathbb{R}^{n_0 \times m}$ and a corresponding set of q-dimensional labels $y = \left[y^{(0)}, \dots, y^{(m)}\right] \in \mathbb{R}^{q \times m}$. As in any regression setting, we assume that $\left(x^{(j)}, y^{(j)}\right)$ are independent and identically distributed draws from an underlying distribution.

The problem consists of finding a prediction rule $f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^q$ from a parametric family of functions indexed by $\theta \in \Theta$, such that the error in terms of the Frobenius norm is minimized. Letting f_θ be applied column-wise to X, the problem can formally be stated as

$$\widehat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{mq} \left\| f_\theta(X) - y \right\|_F^2 = \underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{q} \sum_{i=1}^{q} \frac{1}{m} \sum_{j=1}^{m} \left[\left(f_\theta \left(x^{(j)} \right) \right)_i - y_i^{(j)} \right]^2, \tag{1}$$

where $\| \cdot \|_F$ denotes the Frobenius norm. The above expression can be interpreted as the average of the mean squared errors over all the q output dimensions. Solving the problem is traditionally referred to as minimizing the loss function.

For any $\theta \in \Theta$ the loss given by the above expression must, in the limit, be lower bounded almost surely by the conditional expectation of y with respect to x. To see this, let first $(\mathbf{x}, \mathbf{y})$ be random variables with the same distribution as our samples. Due to the independence assumption it holds for any $\theta \in \Theta$,

$$\frac{1}{m} \sum_{j=1}^{m} \left[f_\theta \left(x^{(j)} \right) - y_i^{(j)} \right]^2 \xrightarrow{a.s.} \mathbb{E} \left[\mathbf{y}_i - f_\theta(\mathbf{x})_i \right]^2 \geqslant \mathbb{E} \left[\mathbf{y}_i - \mathbb{E} \left(\mathbf{y}_i | \mathbf{x} \right) \right]^2,$$

implying that in the limit the loss function is minimized by the map that is the conditional expectation of the label given the feature in each output dimension.

As will become apparent in section 2.3, the parametric family is able to approximate any continuous function in the limit. In particular, it can approximate conditional expectations to arbitrary precision. For m tending to infinity, this implies that the solution to the problem in (1) is solved by the $\widehat{\theta} \in \Theta$ such that

$$\forall x \in \mathbb{R}^q : \quad f_{\widehat{\theta}}(x) = E \left[\mathbf{y} | \mathbf{x} = x \right] = \left(\mathbb{E} \left[y_1 \middle| \mathbf{x} = x \right], \dots, \mathbb{E} \left[y_q \middle| \mathbf{x} = x \right] \right)^\top$$

2.2 Model and Architecture

An ANN is a specific parametric family, $\{f_\theta | \theta \in \Theta\}$, of functions with the property that f_θ can represent any function (in the limit) and that the existence of the backpropagation algorithm allows for an extraordinarily fast calculation of the gradient.

The multi-layer feedforward ANN consists of an input layer and an a priori chosen number of subsequent layers L. We denote the last layer as the output layer and the remaining $L-1$ layers as the hidden layers. Each hidden layer $l \in \{1, \dots, L-1\}$ consists of an a priori chosen number of units n_l and the output layer has $n_L = q$ by necessity. In layer l, all the units are represented as an $n_l \times m$-matrix $a^{[l]}$, where the i'th row corresponds to the i'th node in the layer.

In the perceptron framework of ANNs, each unit is dependent on all former layers through the introduction of the parameters $w^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $b^{[l]} \in \mathbb{R}^{n_l}$, and the relationship

$$\begin{aligned} a^{[0]} &= z^{[0]} = X, \\ a^{[l]} &= g^{[l]} \left(z^{[l]} \right) = g^{[l]} \left(w^{[l]} a^{[l-1]} + b^{[l]} \right), \qquad l \in \{1, \dots, L\} \\ f_\theta(X) &= a^{[L]}, \end{aligned}$$

where $b^{[l]}$ is added column wise to $w^{[l]} a^{[l-1]}$ if necessary. The function $g^{[l]} : \mathbb{R} \to \mathbb{R}$ is traditionally referred to as the activation function, which is applied element wise to $w^{[l]} a^{[l-1]} + b^{[l]}$. Normally, the activation functions applied to the hidden layers are characterized by $g^{[l]} = g$ for some element-wise function g and the output layer has the identity function as activation function. A representation of a feedforward ANN can be seen in figure 2.1

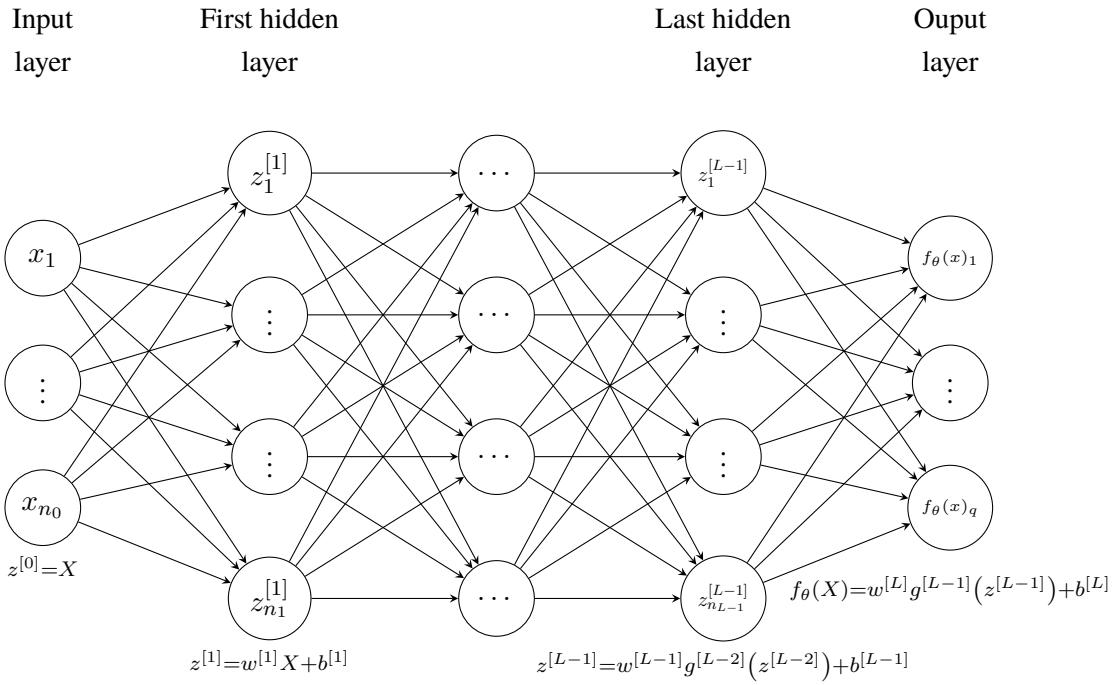


Figure 2.1: Representation of an artificial neural network

The ANN is characterized by all weights $w^{[l]}, b^{[l]}$ and activation functions $g^{[l]}$ for $l = 1, \ldots, L$. The activation functions are fixed and hence not learnable, which is why we only consider θ as representing all the weights $w^{[l]}$ and $b^{[l]}$ for $l = 1, \ldots, L$.

We refer to an evaluation of f_θ as a forward pass of the network. Note that a forward pass can be executed for any number of samples, m. This implies that we can evaluate f_θ for a single sample or for a fraction of the total sample set (see section 2.6 on mini-batch gradient descent). Ignoring activation and addition, and assuming the same number of units in each layer, n: one forward pass of m samples consists of $L - 1$ matrix products between a $n \times n$ and $n \times m$ matrix and a final matrix product between a $q \times n$ and $n \times m$ matrix. Thus the complexity of one forward pass is of the order of $n_0mn + (L - 2)n^2m + nqm$.

The weight matrix $w^{[l]}$ has $n_{l-1}n_l$ parameters and the bias vector $b^{[l]}$ has n_l parameters. Adding these two numbers together for every layer reveals that the number of parameters in an artificial neural network is

$$(n_0 + 1)\, n_1 + \sum_{l=2}^{L-1} (n_{l-1} + 1)\, n_l + (n_{L-1} + 1)\, q. \tag{2}$$

2.3 A Universal Representation Theorem

It is often stated that ANNs are capable of approximating any continuous function due to the universal representation theorem. However, many universal representation theorems exist and most of them are specialized to a specific class of activation functions. When training an ANN in a finance setting, it is crucial that the activation functions are strictly positive and unbounded, since they should emulate the shape of the true pricing functions or implied volatility surfaces. In this section, one such universal representation theorem is presented.

Denote the family of continuous functions defined on $\mathbb{R}^n$ as $C(\mathbb{R}^n)$. These are the types of functions we wish for the ANN to approximate. The set of continuous functions does not constitute a metric space, so we first need to specify what it means to approximate a continuous function.

Let $\mathbb{B}_n$ and λ^n respectively denote the n-dimensional Borel σ-algebra and Lebesgue-measure. A commonly used class of spaces in measure theory is the $\mathcal{L}^p$-spaces for $p \geqslant 1$. The limit of these spaces for some set $X \subseteq \mathbb{R}^n$

as p tends to infinity is [20, ch. 13]

$$\mathcal{L}^{\infty}(X) := \{f : X \longrightarrow \mathbb{R} | f \text{ is } \mathbb{B}_n - \mathbb{B} \text{ measurable}, \exists c > 0, \lambda^n (\{x \in X : |f(x)| \geqslant c\}) = 0\}. \tag{3}$$

The family of interest in this section is the more restrictive family of locally integrable functions $\mathcal{L}^{\infty}_{\text{loc}}$ implicitly defined through the relationship

$$f \in \mathcal{L}^{\infty}_{\text{loc}}(\mathbb{R}^n) \iff \forall K \subset \mathbb{R}^n, K \text{ compact} : f \in \mathcal{L}^{\infty}(K).$$

A subset D of $\mathcal{L}^{\infty}_{\text{loc}}$ is said to be dense in $C(\mathbb{R}^n)$ if for every $u \in C(\mathbb{R}^n)$ and compact subset $K \subset \mathbb{R}^n$, there exists a sequence $(f_k)_{k\in\mathbb{N}} \subset D$ such that

$$\liminf_{k\to\infty} \{c > 0 : \lambda^n (\{|u - f_k| \geqslant c\} \cap K) = 0\} = 0,$$

where λ^n is the n-dimensional Lebesgue-measure. In other words, the function u can be approximated arbitrarily well by functions in D except on null-sets. It is interesting to note that this definition does not require D to consist of continuous functions, as long as they are continuous a.e.

Let σ be a locally integrable activation function. It follows that the map $x \longmapsto \sigma\left(w^{\top}x + b\right)$ for any $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ is locally integrable as well. Assuming that the output activation function is the identity map, it follows that the set of all shallow[1] ANNs is the span of such maps, i.e.

$$D_n := \left\{ x \longmapsto \sum_{i=1}^{k} \lambda_i \sigma\left(w_i^{\top}x + b_i\right) \middle| k \in \mathbb{N}, \lambda_i \in \mathbb{R}, w_i \in \mathbb{R}^n, b \in \mathbb{R} \right\}.$$

Since $\mathcal{L}^{\infty}_{\text{loc}}$ is a vector space it follows that $D_n \subset \mathcal{L}^{\infty}_{\text{loc}}$. The question is what restrictions we need to impose on the activation function, other than local integrability, in order to ensure that D_n is dense in $C(\mathbb{R}^n)$.

It is apparent that we must require that σ is not polynomial. To see this, note that if σ is a polynomial of degree m, then all elements of D_n are polynomials of degree m as well. However, polynomials of a fixed degree cannot approximate any continuous function to arbitrary precision.

It turns out that the only other restriction we need to impose on σ is that the closure of all discontinuity points has Lebesgue-measure zero [17]. Most activation functions used in practice are continuous, so this technicality is unimportant.

We conclude that if an ANN with a continuous, locally integrable and non-polynomial activation function cannot appropriately approximate a given continuous target function, the problem stems from a lack of nodes. However, the result does not examine the number of nodes needed for such an approximation. A rather convincing argument presented in section 2.4 suggests that this number has an exponential nature. The problem is, however, remedied by utilizing deep networks[2].

The activation functions used in this project are the element-wise application of the rectifier and its smooth extension softplus,

$$\max\{x, 0\}, \qquad \ln(1 + e^x).$$

Both of these activation functions trivially satisfies the requirements put forth in this section. For a subset $K \subseteq \mathbb{R}^n$ to be compact, it must be closed and bounded. Since both of these functions are increasing and finite on every closed and bounded set, we simply pick c as the largest element in K. From the definition in (3) it trivially follows that the activation function is an element of $\mathcal{L}^{\infty}(K)$.

2.4 Representation Benefits of Deep Feedforward Networks

From section 2.3, it is apparent that a shallow ANN can under very mild restriction approximate any continuous function to arbitrary precision as long as the network consists of enough units. Conventional wisdom does, however,

[1] An ANN with only one hidden layer.

[2] An ANN with multiple hidden layers.

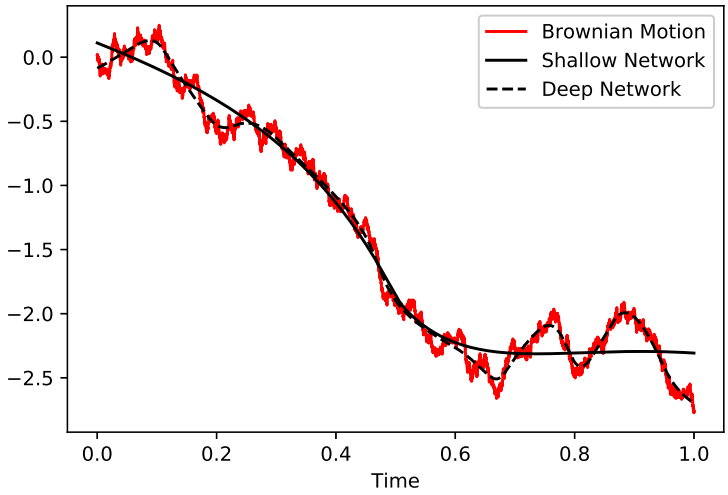


Figure 2.2: An illustration of shallow vs. deep ANNs' ability to approximate a realisation of a Brownian Motion. Both networks consists of 18766 parameters, but they are distributed respectively through a single layer with 6255 units vs. 10 layers with 45 units. Both networks where trained using the methods described in section 2.6.

dictate that ANNs with multiple hidden layers perform better than a simple shallow network. The intuition is that the composition of layers increases the complexity of the network faster than the addition of units does. Hence, a deep network is better equipped to approximate the details of a function. An illustration of this phenomenon is illustrated in figure 2.2. In the figure, it is clear that while both ANNs possesses the same number of parameters, the shallow network is incapable of mimicking the finer details of the highly complex function.

In this section, a detailed overview of the results from [7] is presented. In short, the article presents an example of a classification problem in which deep ANNs outperform the shallow networks. The proofs and results confirm the conventional wisdom stated above. At the end of the section, we demonstrate how the classification results can easily be translated into a regression setting, which is the main focus of this article. This section will consist of the following parts:

1. Introduce the n-alternating-point problem (n-ap).
2. Construct a deep ANN with zero classification error in the n-ap problem.
3. Find a lower bound for the classification error in the n-ap problem as a function of the number of units and layers.
4. Use the results to conclude the superiority of the deep network and extend the conclusion to the regression setting.

Throughout this section the notation of section 2.2 is used. We further introduce the following notation: the class of artificial neural networks with L layers (excluding input layer, including output layer) with at most m units in each hidden layer and activation function $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ used in all layers *including the output layer* is denoted as $\mathfrak{N}(\sigma; m, L)$.

Classification with ANNs is almost identical to regression, but for completeness, a short introduction is presented. The main difference is that the labels in the sample set $((x_i, y_i))_{i=1}^n$ are only allowed to vary between a finite set of values. In this section we only consider binary one-dimensional classification, so $y_i \in \{0, 1\}$.

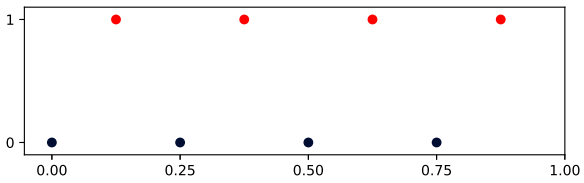


Figure 2.3: The 8-ap problem.

A feature x_i is then classified by an ANN f through the classification map $\widetilde{f}(x) := 1_{\{f(x)\geqslant 1/2\}}$ and the corresponding classification error can be represented by the functional

$$\mathcal{R}(f) := \frac{1}{n}\sum_{i=1}^{n} 1_{\{\widetilde{f}(x_i)\neq y_i\}}.$$

The n-ap problem is a specific zero-indexed sample set $((x_i, y_i))_{i=0}^{n-1}$ used, in this section, to measure the performance of a classification procedure. The set consists of n uniformly spaced features on $\left[0, 1-\frac{1}{n}\right]$ with alternating labels. More formally: for $i \in \{0, \ldots, n-1\}$ define

$$x_i := \frac{i}{n}, \qquad y_i := \begin{cases} 0 & \text{if } i \text{ is even} \\ 1 & \text{if } i \text{ is odd} \end{cases}.$$

The 8-ap problem is depicted in figure 2.3. This problem is intuitively easy to solve. It turns out however, that once the number of units is limited, it becomes very difficult for a shallow network to classify this sample set correctly.

Note that the article [7] is inconsistent in the definition of the n-ap problem. This inconsistency is not present in the following sections.

2.4.1 Construction of the Perfect Network

This section consists of constructing a function that perfectly solves the n-ap problem for $n := 2^k$ for some integer k and prove that the function can be represented by a deep ANN.

The central object in the construction is the mirror map $f_m : \mathbb{R} \longrightarrow \mathbb{R}$, which is defined as

$$f_m(x) := \begin{cases} 2x & 0 \leqslant x \leqslant \frac{1}{2} \\ 2(1-x) & \frac{1}{2} < x \leqslant 1 \\ 0 & \text{otherwise} \end{cases}.$$

Note that the mirror map is continuous which implies that the soft and strict inequalities in the definition can be interchanged at will. We also define the k-composition of the mirror map as $f_m^k(x) = (f_m \circ \cdots \circ f_m)(x)$ depicted in figure 2.4. Lastly, the activation function considered is the rectifier[3]

$$\sigma_R(x) := \max\{x, 0\}.$$

The goal of this section is to show that $f_m^k \in \mathfrak{R}(\sigma_R, 2, 2k)$ and that $f_m^k(x_i) = \widetilde{f_m^k}(x_i) = y_i$ on every (x_i, y_i) in the 2^k-ap.

Observe that we can represent the mirror map as $f_m(x) = \sigma_R(2\sigma_R(x) - 4\sigma_R(x - \frac{1}{2}))$ which implies that the mirror map can be represented by an ANN with one hidden layer i.e. $f_m \in \mathfrak{R}(\sigma_R, 2, 2)$. This further implies that the composite mirror map f_m^k can be viewed as k connected ANNs with two layers (including the last layer); this will itself be an ANN with $2k$ layers with at most two units in each layer. This certainly implies that $f_m^k \in \mathfrak{R}(\sigma_R, 2, 2k)$.

In order to show that the composite mirror map solves the n-ap. problem perfectly, we need to obtain a more rich understanding of f_m^k; the following lemma will do just that.

[3]This activation function is commonly referred to as the ReLU activation function. However, ReLU stands for rectified linear unit and technically refers to a unit in an artificial neural network which utilized the rectifier as activation function.

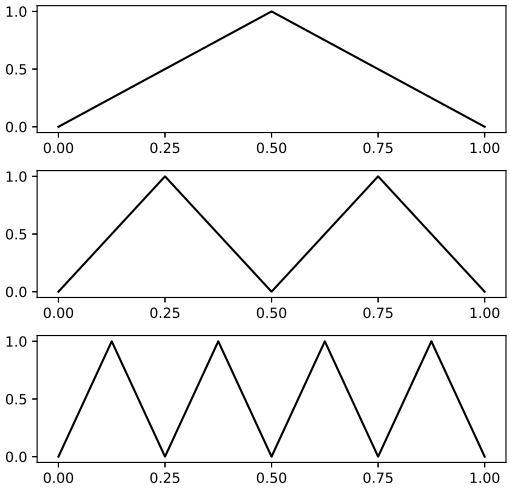


Figure 2.4: f_m, f_m^2 and f_m^3.

Lemma 2.1. *Let $x \in [0,1]$ and positive integer k be given, and choose the unique non-negative integer $i_k \in \{0,1,\ldots,2^{k-1}\}$ and real $x_k \in [0,1)$ so that $x = (i_k + x_k)2^{1-k}$. Then*

$$f_m^k(x) = \begin{cases} 2x_k & 0 \leqslant x_k \leqslant \frac{1}{2} \\ 2(1-x_k) & \frac{1}{2} < x_k < 1 \end{cases}$$

Proof. This proposition is proven to be true for any positive integer k by induction. For $k = 1$ the above is trivially satisfied by definition of the mirror map.

Assume for induction that the above holds for some integer k and that $x \in \left[0, \frac{1}{2}\right]$. By definition of the mirror map we have $f_m^{k+1} = (f_m^k \circ f_m)(x) = f_m^k(2x)$. Now choose $i_k \in \left\{0,1,\ldots,2^{k-1}\right\}$ and $x_k \in [0,1)$ such that $2x = (i_k + x_k)2^{1-k}$ which implies $x = (i_k + x_k)2^{1-(k+1)}$. By induction it holds (for $x_{k+1} = x_k$ and $i_{k+1} = i_k$)

$$f_m^{k+1}(x) = f_m^k(2x) = \begin{cases} 2x_k & 0 \leqslant x_k \leqslant \frac{1}{2} \\ 2(1-x_k) & \frac{1}{2} < x_k < 1 \end{cases} = \begin{cases} 2x_{k+1} & 0 \leq x_{k+1} \leq \frac{1}{2} \\ 2(1-x_{k+1}) & \frac{1}{2} < x_{k+1} < 1 \end{cases}$$

which shows the induction step for $x \in \left[0, \frac{1}{2}\right]$.

Assume for induction that the above holds for some integer k and that $x \in \left(\frac{1}{2}, 1\right]$. Then by definition of the mirror map $f_m^{k+1}(x) = (f_m^k \circ f_m)(x) = f_m^k(2 - 2x)$. Now choose unique $i_k \in \{0,1,\ldots,2^{k-1}\}$ and $x_k \in [0,1]$ such that $2 - 2x = (i_k + x_k)2^{1-k}$ which implies $x = (2^k - i_k - x_k)2^{1-(k+1)}$. Observe that the unique choice of i_{k+1} and x_{k+1} depend on x_k. More specifically,

- If $x_k = 0$ set $i_{k+1} = 2^k - i_k$ and $x_{k+1} = 0$
- If $x_k > 0$ set $i_{k+1} = 2^k - i_k - 1$ and $x_{k+1} = 1 - x_k$

In both cases $i_{k+1} \in \{0, 1, \ldots, 2^{(k+1)-1}\}$, $x_{k+1} \in [0, 1]$ and $x = (i_{k+1} + x_{k+1})2^{1-(k+1)}$ (also uniquely). Using the inductive property of f_m^k we get

$$\begin{aligned} f_m^{k+1}(x) = f_m^k(2 - 2x) &= \begin{cases} 2x_k & 0 \leq x_k \leq \frac{1}{2} \\ 2(1 - x_k) & \frac{1}{2} < x_k < 1 \end{cases} \\ &= \begin{cases} 2x_{k+1} & x_{k+1} = 0 \\ 2(1 - x_{k+1}) & 0 < x_{k+1} \leq \frac{1}{2} \\ 2x_{k+1} & \frac{1}{2} < 1 - x_{k+1} < 1 \end{cases} \\ &= \begin{cases} 2x_{k+1} & 0 \leq x_{k+1} \leq \frac{1}{2} \\ 2(1 - x_{k+1}) & \frac{1}{2} < x_{k+1} < 1 \end{cases} \end{aligned}$$

which shows the induction step for $x \in (\frac{1}{2}, 1]$. □

Using the above lemma we can now show that f_m^k perfectly classifies the 2^k-ap.

Lemma 2.2. *Let $((z_i, y_i))_{i=1}^{n-1}$ be given by the n-ap. for some $n = 2^k$ for positive integer k. Then f_m^k satisfies $\mathcal{R}(f_m^k) = 0$.*

Proof. Assume $((z_i, y_i))_{i=1}^{n-1}$ is given by the n-ap. for some $n = 2^k$ for positive integer k. For $i \in \{0, n-1\}$ the feature can be written as

$$z_i = \begin{cases} \left(\frac{i}{2} + 0\right) 2^{1-k} & \text{if i is even} \\ \left(\frac{i-1}{2} + \frac{1}{2}\right) 2^{1-k} & \text{if i is odd} \end{cases}.$$

In the notation of lemma 2.1 it follows that the helper variable x_k is zero for even i and one half for odd i. From lemma 2.1 we gather that $f_m^k(z_i) = \widetilde{f_m^k}(z_i) = y_i$ for all i, which of course implies a classification error of zero. □

2.4.2 A Lower Bound on the Classification Error

From the universal representation theorem we expect a lower bound on the classification error of some ANN to converge towards zero as the number of units tends to infinity. What still remains unanswered is how fast the lower bound for the n-ap. converges to zero in relation to the number of units and the number of layers. As we shall see in this section: the lower bound decreased linearly in number of units and exponentially in the number of layers.

Finding a lower bound on the classification error boils down to understanding the behavior of t-sawtooth functions. A function $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ is said to be t-sawtooth if it splits $\mathbb{R}$ into t intervals and is linear within each interval. While the set of sawtooth functions is closed under addition and composition, the growth of the complexity is highly dependent on which operation one uses.

Lemma 2.3. *Let $f : \mathbb{R} \longrightarrow \mathbb{R}$ and $g : \mathbb{R} \longrightarrow \mathbb{R}$ be respectively k- and l-sawtooth. Then $f + g$ is at most $(k + l)$-sawtooth and $f \circ g$ is at most kl-sawtooth.*

Proof. Let $\mathcal{I}_f$ denote the set of the k intervals in which f is linear. Let $\mathcal{I}_g$ be defined in the same manner.

Consider first the map $f + g$ and pick $U_f \in \mathcal{I}_f$ and $U_g \in \mathcal{I}_g$ arbitrarily. If $U_f \cap U_g$ is not empty, then $f + g$ constitutes a sum of two linear functions in this interval, which implies $f + g$ is linear in this interval. The real line can be partitioned into a union of these intersections:

$$\bigcup_{U_f \in \mathcal{I}_f} \bigcup_{U_g \in \mathcal{I}_g} U_f \cap U_g = \bigcup_{U_f \in \mathcal{I}_f} \left(U_f \cap \bigcup_{U_g \in \mathcal{I}_g} U_g \right) = \bigcup_{U_f \in \mathcal{I}_f} U_f = \mathbb{R},$$

and there is at most $k + l$ distinct and non-empty intersections. To see the last part, one simply has to realise that $\mathcal{I}_f$ and $\mathcal{I}_g$ contains respectively k and l left end points. Each intersection must contain a left end point from either $\mathcal{I}_f$ or

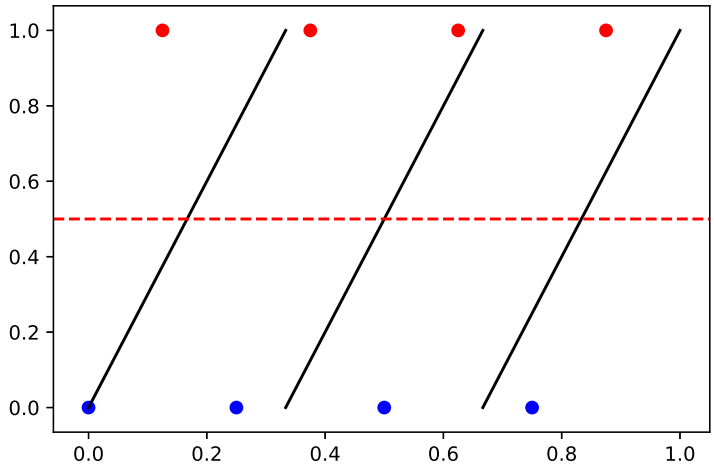


Figure 2.5: An illustration of a t-sawtooth functions crossing one half $2t-1$ times for $t=3$ along with the 8-ap.

$\mathcal{I}_g$ and they cannot be reused. We conclude that we can partition $\mathbb{R}$ into no more than $k+l$ intervals in which $f+g$ is linear, i.e. $f+g$ is at most $(k+l)$-sawtooth.

Consider next the map $f \circ g$ and pick $U_g \in \mathcal{I}_g$. Since g is affine on U_g, the image $g(U_g)$ is an interval. Thus f can at most partition $g(U_g)$ into k intervals. The set $\mathcal{I}_g$ contains l of such intervals, so the map $f \circ g$ can at most partition l intervals into k pieces; it is at most kl-sawtooth. □

The growth in complexity of sawtooth functions is inherited by the ANNs with sawtooth activation functions.

Lemma 2.4. *If σ is t-sawtooth, then every $f \in \mathfrak{N}(\sigma; m, l)$ with $f : \mathbb{R} \longrightarrow \mathbb{R}$ is $(tm)^l$-sawtooth.*

Proof. The proof consists of showing that the output of layer i is $(tm)^i$-sawtooth through induction.

With the notation of section 2.2 the first hidden layer is a composition of the map $x \longmapsto w^{[1]}x + b^{[1]}$ and the activation function. In other words, a composition of a 1-sawtooth and t-sawtooth function. By lemma 2.3, the first layer is t-sawtooth, which is definitely at most tm-sawtooth.

Consider now the i'th layer. By the inductive hypothesis, the input is an at most m-dimensional vector $a^{[i-1]}$ consisting of functions which are at most $(tm)^{i-1}$-sawtooth. The vector is then transformed through the map

$$a^{[i-1]} \longmapsto \sigma\left(w^{[i]}a^{[i-1]} + b^{[i]}\right).$$

In other words a sum of at most m functions which are at most $(tm)^{i-1}$-sawtooth and then a composition with a t-sawtooth activation function. By lemma 2.3 the output is a $(tm)^i$-sawtooth map. □

Recall the classification map $\widetilde{f}(x) = 1_{\{f(x) \geqslant 1/2\}}(x)$ and the corresponding mean classification error $\mathcal{R}$. Since it is proven that any ANN with sawtooth activation function is itself sawtooth, all there is left to prove is a lower bound on the mean classification error for an arbitrary sawtooth function.

Lemma 2.5. *Let $((x_i, y_i))_{i=0}^{n-1}$ be given according to the n-ap. Then every t-sawtooth function $f : \mathbb{R} \longrightarrow \mathbb{R}$ satisfies $\mathcal{R}(f) \geqslant (n-4t)/(3n)$.*

Proof. Since f is affine within each of its partitions, it can only cross one half once in every interval. Furthermore, at each right endpoint except for the last one, the function can cross one half by jumping (see figure 2.5). Thus f can cross one half at most $2t-1$ times.

The classification map $\widetilde{f}$ is then piecewise constant with $2t$ partitions of the real line. The extra partition follows from the interval between the first left endpoint and the first crossing.

The lower bound can then be constructed by considering the best-case scenario. There are n points that are spread out across $2t$ intervals. If two points fall within one interval, it is theoretically possible to classify them perfectly. However, with more than two points, at least one third will be classified incorrectly.

Placing two points in each of the $2t$ intervals leaves us with $n - 4t$ points unaccounted-for. At least one-third of these remaining points will be classified incorrectly since each point will be placed in an interval which already contains at least two points. Since the classification error is an empirical average, we divide this number further by n proving the result. □

2.4.3 Conclusion and Implications of the Result

The content of section 2.4.1 and 2.4.2 can be summarized in the main theorem of article [7].

Theorem 2.6. *Let positive integer k, number of layers L, and number of nodes per layer n be given. Given a t-sawtooth activation function $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ and $n := 2^k$ points as specified by the n-ap, then*

$$\min_{f \in \mathfrak{R}(\sigma_R; 2, 2k)} \mathcal{R}(f) = 0 \quad \textit{and} \quad \min_{g \in \mathfrak{R}(\sigma; m, L)} \mathcal{R}(g) \geqslant \frac{n - 4(tm)^L}{3n}.$$

Proof. The upper bound follow from lemma 2.2 and the fact that $f_m^k \in \mathfrak{R}(\sigma_R, 2, 2k)$. The lower bound follows from a trivial combination of lemma 2.4 and 2.5. □

Conclusions about the regression setting can be drawn from this result. From the proof of 2.2 it is apparent that the composite mirror map does in fact regress the n-ap perfectly, so the error measure from equation (1) is zero as well. Furthermore, a non-zero classification error must imply a non-zero regression error.

The result provides a strong argument in favor of depth in ANNs, but there are at least three caveats that are worth mentioning. The first issue arises from the sawtooth assumption. This assumption was crucial in the derivation of the lower bound, but the activation functions used in regression are rarely sawtooth. While they often are monotone, it is not clear how well the behavior of sawtooth functions carries over to non-linear monotone functions.

The second issue is more subtle. When comparing two ANNs' ability to reduce training error, one must equalize the number of parameters since one would otherwise always expect the model with the highest number of parameters to outperform. Consider a network with the rectifier activation function and $m = 2^{(k-3)/L-1}$ units in each layer. From theorem 2.6 the lower bound can be found as

$$\min_{g \in \mathfrak{R}(\sigma; m, L)} \mathcal{R}(g) \geqslant \frac{n - 4(2m)^L}{3n} = \frac{1}{3} - (2m)^L 2^{-k} \left(\frac{4}{3}\right) = \frac{1}{6}.$$

The formula from equation (2) yields the number of parameters for respectively the shallow ($L = 2$) and deep ($m = 2$ and $L = 2k$) network to be

$$3 \cdot 2^{\frac{k-5}{2}} + 1 \quad \text{and} \quad 12k - 5.$$

The number of parameters in the shallow network is smaller than the number of parameters in the deep network for all $k \leqslant 17$. Thus, the deep network can only be said to outperform the shallow network once the number of points in the n-ap surpasses approximately a quarter million.

Lastly, only the n-ap problem is considered, which turns out to favor deep ANNs due to their capability to increase complexity fast. In this project, problems relating to finance applications are considered. It is not entirely clear whether these results are transferable to a finance setting. An attempt to answer this is the central theme of section 3.

2.5 The Backpropagation Algorithm

The problem described in equation (1) is a non-convex problem without a known analytical solution. A gradient descent algorithm is, therefore, the natural choice as a numerical training procedure. Since these types of algorithms

only require computation of the first derivative and the parametric family is specifically designed to allow for fast computation of the gradient, the algorithm should converge relatively fast towards a local minimum. The backpropagation algorithm is a procedure utilizing the iterative structure of the ANN to compute the gradient effectively.

Before deriving the gradient, we assume that a feedforward pass of the neural network is completed where all $a^{[l]}$ and $z^{[l]}$ for $l = 1, \dots, L$ is saved. It will become apparent through the application of the chain rule that fast derivation of the gradients requires knowledge of these particular intermediate states.

We wish to derive the backpropagation algorithm using the full set of observations

$$X = \left[x^{(1)}, \dots, x^{(m)}\right] \in \mathbb{R}^{n_0 \times m}, \qquad y = \left(y^{(1)}, \dots, y^{(m)}\right) \in \mathbb{R}^{q \times m}$$

The mean square error loss function then becomes $c = \frac{1}{mq} \left\|\hat{y} - y\right\|_F^2$, where $\hat{y} = f_\theta(X)$ is the model prediction of y given X. The first gradient we wish to determine is with respect to $\hat{y} = a^{[L]}$, which is fairly simple

$$\frac{\partial c}{\partial a^{[L]}} = \frac{\partial c}{\partial \hat{y}} = \left[\frac{\partial c}{\partial \hat{y}_{i,j}}\right]_{i,j} = \left[\frac{2}{mq}(\hat{y}_{i,j} - y_{i,j})\right]_{i,j} = \frac{2}{mq}(\hat{y} - y) \in \mathbb{R}^{q \times m} \tag{4}$$

Given that we know $\frac{\partial c}{\partial a^{[l]}}$ for $l \in \{1, \dots, L\}$ it is possible to determine $\frac{\partial c}{\partial z^{[l]}}$. We start by focusing on finding the derivative of c with respect to each element in $z^{[l]}$

$$\frac{\partial c\left(g^{[l]}\left(z^{[l]}\right)\right)}{\partial z_{i,j}^{[l]}} = \frac{\partial c\left(g^{[l]}\left(z_{1,1}^{[l]}\right), g^{[l]}\left(z_{1,2}^{[l]}\right), \dots, g^{[l]}\left(z_{n_l,m}^{[l]}\right)\right)}{\partial z_{i,j}^{[l]}} = g'^{[l]}\left(z_{i,j}^{[l]}\right) \frac{\partial c}{\partial a_{i,j}^{[l]}}$$

Here we view c as a function that calculates the empirical loss from $a^{[l]} = g^{[l]}\left(z^{[l]}\right)$. We also choose to view c as a function of $n_l m$ variables instead of a $n_l \times m$ matrix, which is equivalent. We should also note that we have used that g is applied element wise to $z^{[l]}$. Knowing the derivative of c with respect to each $z_{i,j}^{[l]}$ now yields us the gradient

$$\frac{\partial c}{\partial z^{[l]}} = \left[\frac{\partial c}{\partial z_{i,j}^{[l]}}\right]_{i,j} = \left[g^{[l]}\left(z_{i,j}^{[l]}\right) \frac{\partial c}{\partial a_{i,j}^{[l]}}\right]_{i,j} = g'^{[l]}\left(z^{[l]}\right) \otimes \frac{\partial c}{\partial a^{[l]}} \in \mathbb{R}^{n_l \times m} \tag{5}$$

where we use $\otimes$ to denote the element wise product. The derivative of c wrt. $z^{[l]}$ is dependent of the derivative of c wrt. $a^{[l]}$, but this was assumed to already be known, which means that we are able to calculate gradient wrt. $z^{[l]}$.

Working further back through the network we assume that $\frac{\partial c}{\partial z^{[l]}}$ is known. We can use this and the fact that $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ to calculate the derivatives of c wrt. $w^{[l]}$, $b^{[l]}$ and finally $a^{[l-1]}$, which will enable us to apply the gradient formulas recursively through the network.

Starting with $w^{[l]}$ and focusing on a single element we see that

$$\frac{\partial c}{\partial w_{i,j}^{[l]}} = \frac{\partial c\left(z_{1,1}^{[l]}\left(w_{i,j}^{[l]}\right), \dots, z_{n_l,m}^{[l]}\left(w_{i,j}^{[l]}\right)\right)}{\partial w_{i,j}^{[l]}} = \sum_{h,k=1}^{n_l,m} \frac{\partial z_{h,k}^{[l]}\left(w_{i,j}^{[l]}\right)}{\partial w_{i,j}^{[l]}} \frac{\partial c}{\partial z_{h,k}^{[l]}} = \sum_{k=1}^{m} a_{j,k}^{[l-1]} \frac{\partial c}{\partial z_{i,k}^{[l]}}$$

We have derived the above using the following: Firstly we have chosen to view c as a function of $z^{[l]}$ through all elements in $z^{[l]}$. Secondly we have chosen to view each element $z_{i,j}^{[l]}$ as a function of $w_{i,j}^{[l]}$, which is appropriate since we are investigating the partial derivative wrt. $w_{i,j}^{[l]}$. Thirdly we have applied the multivariate chain rule and fourthly we have used that

$$\frac{\partial z_{h,k}^{[l]}\left(w_{i,j}^{[l]}\right)}{\partial w_{i,j}^{[l]}} = \begin{cases} a_{j,k}^{[l-1]} & h = i \\ 0 & h \neq i \end{cases}$$

which is easily seen when considering $z_{h,k}^{[l]} = \sum_{t=1}^{n_{l-1}} \left(w_{h,t}^{[l]} a_{t,k}^{[l-1]}\right) + b_h^{[l]}$. Knowing the derivative of c with respect to each $w_{i,j}^{[l]}$ now yields us the gradient

$$\frac{\partial c}{\partial w^{[l]}} = \left[\frac{\partial c}{\partial w_{i,j}^{[l]}}\right]_{i,j} = \left[\sum_{k=1}^{m} a_{j,k}^{[l-1]} \frac{\partial c}{\partial z_{i,k}^{[l]}}\right]_{i,j} = \frac{\partial c}{\partial z^{[l]}} a^{[l-1]\top} \in \mathbb{R}^{n_l \times n_{l-1}} \tag{6}$$

which we are able to determine since we know $a^{[l-1]}$ from the forward pass and we know $\frac{\partial c}{\partial z^{[l]}}$ from equation (5).

We wish now to determine $\frac{\partial c}{\partial b^{[l]}}$ using the same methods as the derivative wrt. $w^{[l]}$. We start by calculating the derivative element wise

$$\frac{\partial c}{\partial b_i^{[l]}} = \frac{\partial c\left(z_{1,1}^{[l]}\left(b_i^{[l]}\right), \ldots, z_{n_l,m}^{[l]}\left(b_i^{[l]}\right)\right)}{\partial b_i^{[l]}} = \sum_{h,k=1}^{n_l,m} \frac{\partial z_{h,k}^{[l]}\left(b_i^{[l]}\right)}{\partial b_i^{[l]}} \frac{\partial c}{\partial z_{h,k}^{[l]}} = \sum_{k=1}^{m} \frac{\partial c}{\partial z_{i,k}^{[l]}}$$

Where like with $w^{[l]}$ have chosen to view c as a function of each element of the matrix $z^{[l]}$ and chosen to view $z_i j^{[l]}$ as a function of $b_i^{[l]}$. As mentioned we also applied the multivariate chain rule using

$$\frac{\partial z_{h,k}^{[l]}\left(b_i^{[l]}\right)}{\partial b_i^{[l]}} = \begin{cases} 1 & h = i \\ 0 & h \neq i \end{cases}$$

again using $z_{h,k}^{[l]} = \sum_{t=1}^{n_{l-1}} \left(w_{h,t}^{[l]} a_{t,k}^{[l-1]}\right) + b_h^{[l]}$. Knowing the derivative of c wrt. each $b_i^{[l]}$ yields us

$$\frac{\partial c}{\partial b^{[l]}} = \left[\frac{\partial c}{\partial b_i^{[l]}}\right]_{i=1,\ldots,m} = \left[\sum_{k=1}^{m} \frac{\partial c}{\partial z_{i,k}^{[l]}}\right]_{i=1,\ldots,m} = \frac{\partial c}{\partial z^{[l]}} 1_m \in \mathbb{R}^{n_l} \tag{7}$$

All of the above relied on the fact that $\frac{\partial c}{\partial a^{[l]}}$ is known. We now wish to show that we can determine $\frac{\partial c}{\partial a^{[l-1]}}$ from the above, which is enough to be able to calculate all gradients recursively starting from $l = L$ to $l = 1$. We start by determining the derivative of c wrt. $a^{[l-1]}$ for each element

$$\frac{\partial c}{\partial a_{i,j}^{[l-1]}} = \frac{\partial c\left(z_{1,1}^{[l]}\left(a_{i,j}^{[l-1]}\right), \ldots, z_{n_l,m}^{[l]}\left(a_{i,j}^{[l-1]}\right)\right)}{\partial a_{i,j}^{[l-1]}} = \sum_{h,k=1}^{n_l,m} \frac{\partial z_{h,k}^{[l]}\left(a_{i,j}^{[l-1]}\right)}{\partial a_{i,j}^{[l-1]}} \frac{\partial c}{z_{h,k}^{[l]}} = \sum_{h=1}^{n_l} w_{h,i}^{[l]} \frac{\partial c}{\partial z_{h,j}}$$

Where we have chosen to view c as a function of each element in $z^{[l]}$ and $z_{h,k}^{[l]}$ as a function of $a_{i,j}^{[l-1]}$. We also used that

$$\frac{\partial z_{h,k}^{[l]}\left(b_i^{[l]}\right)}{\partial a_{i,j}^{[l]}} = \begin{cases} w_{h,i} & k = j \\ 0 & k \neq j \end{cases}$$

Knowing the derivative of c wrt. each $a_{i,j}^{[l]}$ yields us

$$\frac{\partial c}{\partial a^{[l-1]}} = \left[\frac{\partial c}{\partial a_{i,j}^{[l-1]}}\right]_{i,j} = \left[\sum_{h=1}^{n_l} w_{h,i}^{[l]} \frac{\partial c}{\partial z_{h,j}}\right]_{i,j} = {w^{[l]}}^\top \frac{\partial c}{\partial z^{[l]}} \in \mathbb{R}^{n_{l-1} \times m} \tag{8}$$

We have now derived equations for all gradients necessary to determine all derivatives of c with respect to $\hat{y} = a^{[L]}$ and $z^{[l]}$, $w^{[l]}$, $b^{[l]}$ and $a^{[l-1]}$ for all $l = L$ to $l = 1$. Using equation (4), (5), (6), (7) and (8) we can summarize the elements in the backpropagation algorithm

$$\begin{aligned}
\frac{\partial c}{\partial a^{[L]}} &= \frac{\partial c}{\partial \hat{y}} = \frac{2}{m}(\hat{y} - y) \in \mathbb{R}^{1 \times m} \\
\frac{\partial c}{\partial z^{[l]}} &= g'^{[l]}\left(z^{[l]}\right) \otimes \frac{\partial c}{\partial a^{[l]}} \in \mathbb{R}^{n_l \times m} \\
\frac{\partial c}{\partial w^{[l]}} &= \frac{\partial c}{\partial z^{[l]}} {a^{[l-1]}}^\top \in \mathbb{R}^{n_l \times n_{l-1}} \\
\frac{\partial c}{\partial b^{[l]}} &= \frac{\partial c}{\partial z^{[l]}} 1_m \in \mathbb{R}^{n_l} \\
\frac{\partial c}{\partial a^{[l-1]}} &= {w^{[l]}}^\top \frac{\partial c}{\partial z^{[l]}} \in \mathbb{R}^{n_{l-1} \times m}
\end{aligned}$$

As already mentioned we need to have completed a forward pass through the model in order to obtain all $z^{[l]}$ and $a^{[l-1]}$ for $l = 1, \ldots, L$, which can then be used to calculate all of the above gradients with a backwards pass through

the model. This is in its essence the backpropagation method for feedforward ANNs.

The backpropagation algorithm allows for quick derivation of all gradients in the neural network. This makes it possible to deploy gradient decent algorithms; Starting from a small non-zero guess w_0 and b_0, we can tune the guess through the iterative rule

$$b_{n+1}^{[l]} \longleftarrow b_n^{[l]} - \gamma_n \frac{\partial c}{\partial b_n^{[l]}},$$
$$w_{n+1}^{[l]} \longleftarrow w_n^{[l]} - \gamma_n \frac{\partial c}{\partial w_n^{[l]}}$$

for $n \geq 0$ where $\gamma_n > 0$ is an appropriately chosen sequence of learning rates. Several improvements to this simple gradient decent algorithm have been suggested. In this article the ADAM algorithm is used [5].

2.6 Training in Practice

Several methods used in order to optimize the speed and performance in the training phase of artificial neural networks are available. This section will provide an overview of the optimizations used throughout this project.

Normalization

If the average value across the training set for a single feature is significantly larger than the other features, then this parameter will influence the gradient the most resulting in this particular feature being prioritized at each iteration. By similar arguments, if the labels are biased, then the gradient will be biased towards a specific label. A biased gradient slows down the speed of convergence.

Another problem arises from the shape of the activation functions. The activation functions used in section 3 and section 4 is the rectifier's smooth extension, softplus,

$$\ln\left(1 + e^x\right).$$

Both functions are asymptotically linear as x tends away from zero. If the activation functions are linear, then so is the output of the network. If the labels are non-linear, then layers of the ANN are used in order to pull the data towards zero in order to produce a non-linear output. This is inefficient both in terms of training speed and in terms of the use of weights.

The problems can be mitigated by normalizing the features and labels[18]. Two methods of normalization are employed in this project. If some feature x has a natural upper and lower bound x_{min} and x_{max}, then a natural normalisation is to scale the feature between -1 and 1 through the formula

$$\frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}.$$

If bounds are not available, the empirical mean and variance of each feature across the training set is computed. Each feature can then be scaled to mean zero and unit variance.

A common critique of both normalization methods is that they are both sensitive to outliers in the data. This is not an issue in this project since all data is constructed artificially.

Mini-batch Gradient Descent

While the calculation of the gradient is fast due to the backpropagation algorithm, it does require a significant amount of memory, if the size of the training set is large. Fortunately, convergence is guaranteed as long as an unbiased estimator of the gradient is used [10]. Thus it is possible to split the training set into mini-batches and compute the gradient of each sample. Since the weights are moved after each calculation of a gradient, the convergence tends to be faster and it is thought to have a regularizing effect. Once the gradient has been computed on every batch, it is said that an entire *epoch* has run.

Increasing the batch size affects training performance in two ways. A higher batch size increases the accuracy of the gradient, but the gradient will be updated less frequently.

In this project, all data is simulated, so the sample size can be controlled at will. If training is to be conducted within a fixed time interval, one has to balance the effects of the sample size and the batch size, since they both increase accuracy at the cost of increased computational time. In most literature, a batch size of 32 is recommended [6], but in this article, we will, in cases, utilize a batch size of 1024, which allows for a larger sample size without slowing down training.

Variance Scaling

The problem in (1) is not convex, so the initial guess determines which local minima the algorithm will converge towards. If all the weights are initialized to zero, then the gradient will be zero as well, resulting in no convergence at all. A common heuristic is to use variance scaling. That is set $b^{[l]} = 0$ and generate random weights $w_{i,j}^{[l]}$ from a mean zero and $1/n^{[l-1]}$ variance normal distribution.

The One-Cycle Policy

Choosing an appropriate sequence of learning rates is crucial for a good fit. Choosing learning rates too high result in divergence of the loss, while choosing them too small results in too slow convergence.

A popular method later referred to as the one-cycle policy for choosing learning rates is proposed by [19]. A number of variants to the approach exist, but they all entail choosing an upper and lower bound for the learning rates. Say the training procedure is terminated after one hundred epochs. One variant of the one-cycle policy then dictates increasing the learning rate linearly from the lower bound to the upper bound and peak at epoch 25. The learning rate will further be decreased linearly and bottom out at epoch 75 and stay there until epoch 100.

A heuristic method for choosing the bounds are also proposed in [19]. Starting from a minimum learning rate of say 10^{-4}, the learning rate is increased after each epoch. The loss should decrease for some time until it eventually explodes. The upper bound is then chosen slightly below the learning rate, which was followed by an explosion in the loss. Finally, the lower bound is chosen a magnitude or two below the upper bound.

Online and Offline Procedures

From a practical point of view, there is an important distinction between online and offline procedures. That is, if the information is gathered in real-time and a subsequent training procedure is to be conducted on the basis of this gathering, we say that the training procedure must be conducted online. An example of this is the need to calculate greeks in real-time for an updated model calibration or trading book. The demand for computational speed is high in this setting.

The opposite is when a procedure is conducted offline. An example of this is when the training of an ANN encompasses a sufficiently wide range of market parameters and thus can be trained at any time. The quality of an offline procedure is not based on the computational speed, but on some other metric, for example, its ability to encompass all market parameters sufficiently.

The analysis of section 3 is purely concerned with online procedures, while section 4 contains both offline and online procedures. This distinction largely explains the differences in the specific choices of hyperparameters.

3 Estimation of Option Prices and Greeks

In this section, we wish to investigate the performance and ability of feed-forward ANNs to estimate the prices of options given market states. We saw in section 2.3 that an ANN, if trained properly, will approximate the conditional expectation of the labels giving the corresponding features.

If we consider a Markovian model with zero interest rate, the price of an option can because of risk-neutral pricing be represented as a conditional expectation of the payoff given a market state. This implies that an ANN

could learn the price of an option as a function of the market state if the network was trained on simulated payoffs and corresponding markets states.

Note the price will only be valid for a given option and time of exposure, which means that training and data generation is done online (see section 2.6), implying that training and data generation should be fast, but still yield precise models. Precision should be both in terms of estimated option prices and derivatives w.r.t. market states. In this article, we focus mostly on measuring and comparing precision and training and not training time. That said, all models in this section have training times between five and 15 seconds on a modest CPU (not even a GPU), including data generation.

This section will consist of four parts:

1. Introduction of ANNs for option pricing, including data generation and performance measuring.
2. Comparison of model performances between ANNs of different depths, but with a comparable number of layers.
3. Introduction of differential ANNs and comparison between ordinary ANNs and differential ANNs.
4. Analysis of the performance of ANNs when compared to ordinary Monte Carlo simulations.

Every analysis in this section will be conducted in the Black-Scholes model in one dimension. This is done to simplify the problem, which will allow us to focus on the ANNs themselves. It is clear that analysis of ANNs in different models with multiple dimensions and for complex options are necessary for a complete understanding of ANNs as a tool for option pricing in practice. It will, however, become clear that the simple setting considered in this section is enough to obtain interesting results.

In this section, we will apply the training procedure described in section 2.6. The models are trained for 100 epochs using a mini-batch size of 1,024 and a cyclic learning rate with a minimum learning rate of 10^{-6} and a varying maximum learning rate.

3.1 Learning Call Option Prices in Black-Scholes

For simplicity, we consider one of the most simple examples, that is estimating call-option prices in the Black-Scholes model. With zero interest rate the model is characterized by the following dynamics for the risky asset under the martingale measure

$$dS(t) = \sigma S(t) dW^Q(t)$$

for some $\sigma > 0$. Our goal is for our network to approximate the price of a European call-option with fixed strike, maturity and volatility (K, T and σ respectively) given the state of the market, i.e., $S(T_{EX}) = s$ at some fixed exposure time, $T_{EX} < T$. For simplicity we will think of the exposure time, T_{EX} as 0 even though we normally do not consider $S(0)$ as a stochastic variable. This does however not have any theoretical or practical consequences. Our goal is, therefore, to estimate $f : \mathbb{R} \to \mathbb{R}$ with

$$f(s) = C^{BS}(s, K, \sigma, T)$$

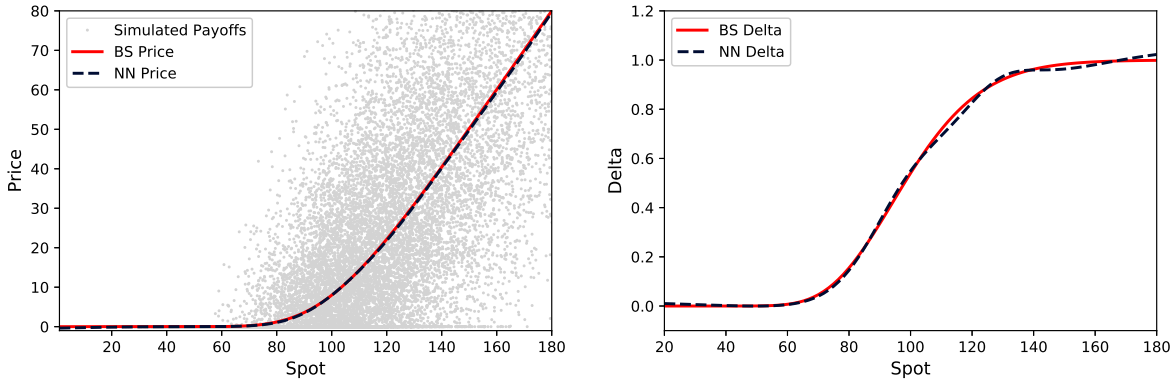
We should note that $f(s)$ is a continuous function in s, which makes it possible to be approximated by an appropriate ANN given enough samples (see section 2.3).

The sample data consists of different spot values as features and their corresponding simulated payoffs as labels, i.e.,

$$X = [S_1(0), \dots, S_m(0)], \quad y = \left[(S_1(T) - K)^+, \dots, (S_m(T) - K)^+\right]$$

Simulating the labels through $S(T)$ requires very little effort in the Black-Scholes model, since $S(T)$ is log-normally distributed [4, p. 69] such that

$$S(T) \overset{d}{=} S(0) \exp\left(-\frac{\sigma^2}{2}T + \sigma\sqrt{T}Z\right) \tag{9}$$



(a) Fit of call-option prices

(b) Fit of call-options delta

Figure 3.1: Fitted call option prices (a) and corresponding delta (b) in the Black-Scholes model with $K = 100$, $T = 1$ and $\sigma = 0.2$ using $m = 30,000$ samples. The model has four layers with five units in each and was trained for 100 epoch using mini-batches of size 1024 and a cyclic learning rate with minimum and maximum learning rate of 10^{-6} and 0.1, respectively

where $Z \sim N(0,1)$. So all that is left, in regards to data generation, is to decide which values of $S(0)$ to consider. Simple approaches consists of choosing $S(0)$ equidistantly from some interval or drawing $S(0)$ uniformly from the same interval. These methods do, however, not consider the fact that some input ranges are harder to approximate than others.

In the case of the price of a call option, the function is linear for extreme spots, so this should be the easiest part to approximate. The second derivative of the option is maximized around the strike, so this should be the most difficult part to approximate. We would, therefore, like to concentrate our input values, $S(0)$, around the strike. We choose to do this through a simulation of log-normal distributed variables,

$$S(0) \overset{d}{=} K_0 \exp\left(-\frac{\sigma_0}{2}T + \sigma_0\sqrt{T}Z\right)$$

where $Z \sim N(0,1)$ and K_0 and σ_0 are chosen depending on the specific call option. When considering a call option with strike $K = 100$ then we choose $K_0 = 100$ and $\sigma_0 = 0.4$. We choose σ_0 high to widen the range of market states that we simulate. This will have the effect of training the ANN to estimate the prices of the call option for a wide range of market states. In section 3.4, the effect of choosing σ_0 low is shown.

As proof of concept, we first choose to fit the price of a call option with strike, $K = 100$ and maturity, $T = 1$ in a model with volatility, $\sigma = 0.2$ using a neural network with four hidden layers and five units in each layer trained on $m = 30000$ samples. We will follow the training procedure described in section 2.6 using mini-batches of size 1024 and a cyclic learning rate with minimum and maximum learning rate of 10^{-6} and 0.1, respectively.

The result can be seen in figure 3.1. Data generation and training was completed in less than 10 seconds. Figure 3.1a shows that the model successfully approximates $C^{BS}(s, 100, 0.2, 1)$. From figure 3.1b, we see that the model also approximated the delta quite well, albeit not perfectly. The deltas of the network are calculated using the backpropagation algorithm and corrected for normalization.

Three error measures are used to estimate the performance of the ANN

1. Measuring the mean square error on the sample data, i.e., equation (1). This measure represents how well the ANN has managed to minimize the loss function. It is, however, not possible to tell if the model is overfitting with this measure. In this article, we refer to this measure as the mean square error on the sample data or **sample MSE**.

2. Measuring the mean square error of the difference between the true price and model predictions from values of s on an equidistant grid from $s = 20$ to $s = 180$ with a thousand points. This measure represents the actual error between the approximation and true price. In this article, we refer to this measure as the mean square error on the true price or **BS MSE**.

3. Measuring the mean square error of the difference between the true delta and first derivative of the model predictions from values of s on an equidistant grid from $s = 20$ to $s = 180$ with a thousand points. In this article, we refer to this measure as the mean square error on the true delta or **BS Delta MSE**.

In figure 3.1, the mean square error is 317.807 on the sample data, 0.0851497 on the true price and 0.000145 on the true delta. We can, furthermore, consider the sample MSE of the true price, i.e., the training loss or sample MSE the model would obtain if it approximated the pricing function perfectly. This measure gives us an idea of the magnitude of sample MSE we can expect from a good model. We will refer to this measure as **BS sample MSE**. For the generated data, the BS sample MSE is 317.292.

The sample MSE and BS sample MSE are observed to be close, which implies that the training of the ANN was successful. If the sample MSE was much higher than BS sample MSE, then we would not expect the ANN to imitate the true price, and if sample MSE was much lower than BS Sample MSE, then we expect the ANN to have badly overfitted the data.

It is hard to conclude anything from BS MSE and BS Delta MSE without comparing these measures for those of other trained ANNs. These measures are best used for comparison between models; however, in section 3.4, we investigate the performance of the ANN for fixed spots when compared to Monte Carlo simulations.

3.2 Analysis of Depth in ANNs for Option Pricing

In section 2.4, we saw that deep ANNs had advantages in some problems over shallow ANNs. A natural question is: Is this also true for an ANN that has to learn option prices from simulated payoffs? In this article, we compare three different models with a comparable number of parameters. The models consist of one, four and nine layers with 35, five and three units per layer, respectively. We will reference these model designs as '1x35', '4x5' and '9x3'. These model designs all have 106 parameters each (see equation 2).

It is not straightforward how one should compare the performance of the above-chosen models. The first issue is that the option price function $C^{BS}(s, 100, 0.2, 1)$ is a very simple function and resembles our choice of activation function (softplus). It might, therefore, be difficult to compare model performances, since we expect all model designs to perform well on this option. We, therefore, choose to compare the model performances on the approximation of the price of a call option with low volatility, $\sigma = 0.1$, and time to maturity, $T = 0.1$, i.e., $C^{BS}(s, 100, 0.1, 0.1)$. This price function has a more pronounced kink (high convexity) around the strike, which increases the complexity of the approximation. However, the simulated payoffs will also have a much lower variance, which makes it easier for the model to learn the price from the payoffs.

The second issue is that if one fits the same model multiple times, then the performance might vary a lot (relatively speaking) due to the random initialization of weights. This variation in performances implies that a model comparison becomes a statistical analysis of average performances of the different model designs. The difficulty arises from the fact that standard variations of model performances, whether measured using sample data or the target function, have a large standard deviation when compared to the mean. One reason for this is that some models do not converge probably or find an inferior local minimum. Since we are not too worried about overfitting, it would be natural to disregard the inferior model fits based on the training loss. It is, however, not obvious how we could do this. Three natural options are

1. Fit m models and disregard the 20% worst performing models measured on the training loss. This method removes inferior models that have not converged to a proper local minimum, but it introduces dependency in the data, which is not desirable.

2. Fit 3 models and keep the only the best performing model measured on training error and repeat this m times to obtain m independent models. This approach mostly eliminates inferior models, but it has the side effect of

Model design	sample MSE	BS MSE
1x35	8.584 (0.110)	0.494 (0.03)
4x5	8.295 (0.178)	0.025 (0.369)
9x3	8.319 (0.178)	0.036 (0.086)

Table 3.1: Lowest sample MSE and BS MSE from figure 3.2 and 3.3 and corresponding learning rate (in parentheses).

turning the statistical analysis into a comparison of the far left tails of the distribution of model performances, which is also not desirable.

3. Fit 5 models and average the performances of the three best models when measured on training loss. Repeat this m times. The m averages are independent and represent a larger part of the distribution of performances than the best model.

In this article, we choose to compare the models using the third option (with $m = 20$) since it produces independent performance data, but without disregarding too many average models.

The third issue is that the optimal learning rate for each model is likely not to be the same. A simple way to remedy this is to repeat the experiment for a range of learning rates for each model and compare the performances from the optimal ranges of each model. In this article, we choose to vary the maximum learning rate of the cycle and keep the minimum learning rate constant at 10^{-6}. For this experiment, we consider 15 maximum learning rates ranging from 0.02 to 0.6, which are spread out equally in log-space to obtain a more even distribution of learning rates across different magnitudes.

To recap: For each model design and each of the 15 maximum learning rates, we train 100 models to obtain 20 performance averages, which are averages of the best three models from groups of five.

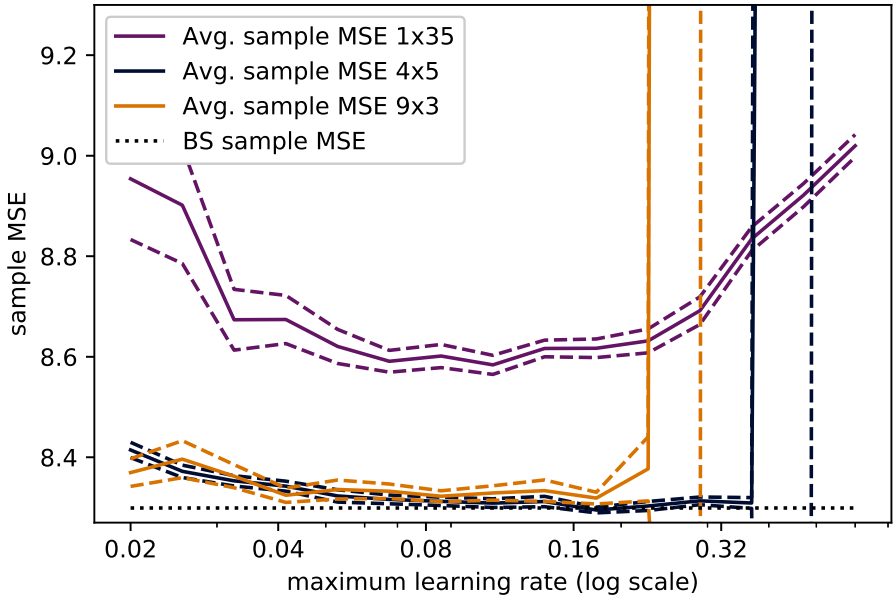
The results can be seen in figure 3.2 showing average sample MSE and figure 3.3 showing average BS MSE. Subfigures labeled (a) has average mean square errors for all model designs where subfigures labeled (b) only has model designs 4x5 and 9x3. Here we see that the optimal maximum learning rate, indeed, is different for the different model designs. The lowest average sample MSE and BS MSE and corresponding learning rates can be seen in table 3.1.

Since the estimates are based on averages of model performances, we can apply the Central Limit Theorem. We can, therefore, utilize quantiles of the normal distribution to estimate confidence bands for the model performances. Looking at the 95% confidence bands, we can also observe that the sample MSE and BS MSE are significantly higher for the 1x35 model design when compared to the 4x5 and 9x3 model designs. There is, however, not a significant difference between the 4x5 and 9x3 model designs, but one can observe that the optimal range of the maximum learning rate is wider for the 4x5 and 1x35 model when compared to the 9x3 model. The 9x3 model design also seems to exhibit higher variance in the trained models when compared to the 4x5 model. These differences suggest that shallower ANNs are easier to train than deep models. For these reasons, we would prefer the 4x5 over the 1x35 and 9x3 model designs, which is also in alignment with [3]. This suggests there is a trade-off between depth and stability/trainability.

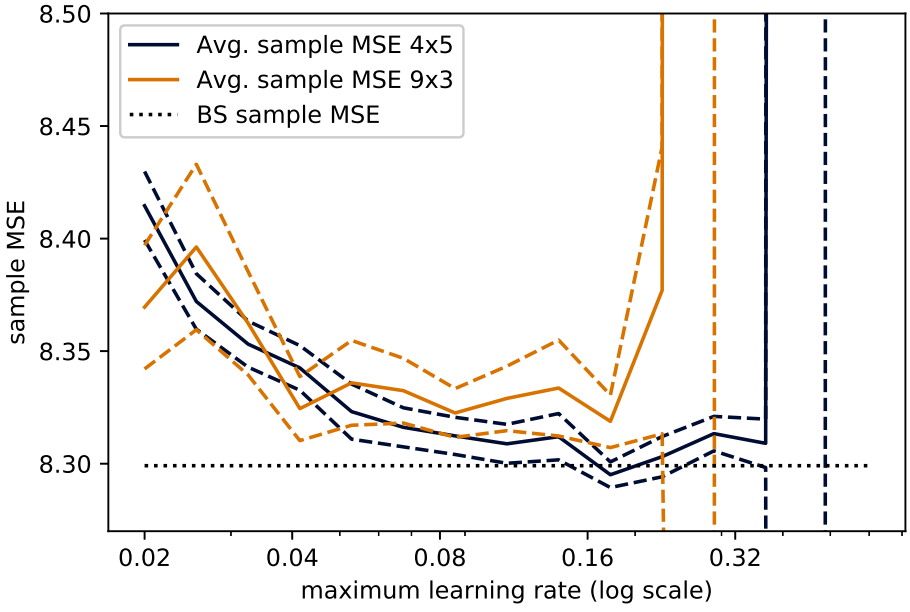
3.3 Differential Neural Networks

Fits of the simple ANN to the Black-Scholes call option price using payoffs are impressive, but figure 3.1a reveals that a great fit of a function does not guaranty a perfect fit of its derivative. In this section, we wish to remedy this.

We will extend our standard neural network model to include a backward pass of the standard model in the forward pass. This is called a differential ANN [3]. A differential ANN with two ordinary hidden layers and two differential layers is depicted in figure 3.4. The hidden differential layers are simply derived using the

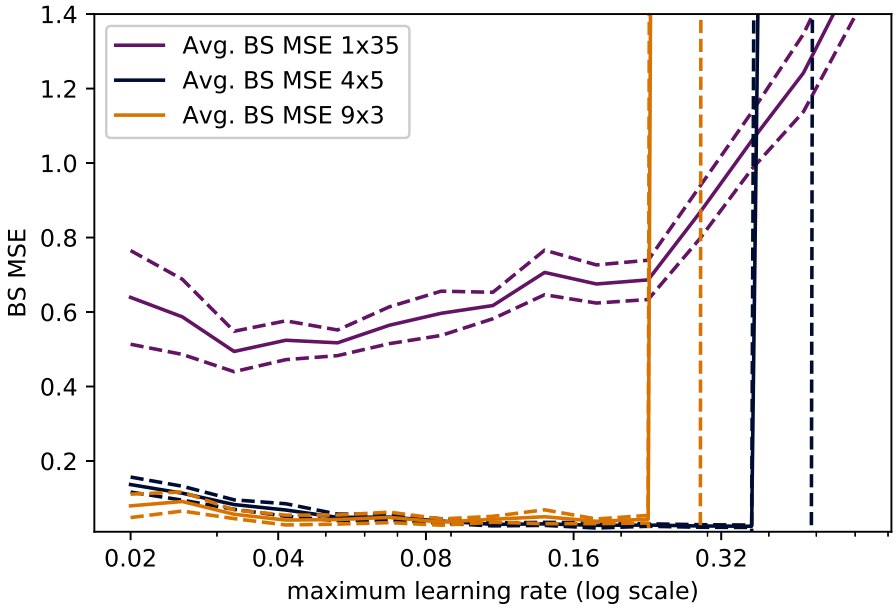


(a)

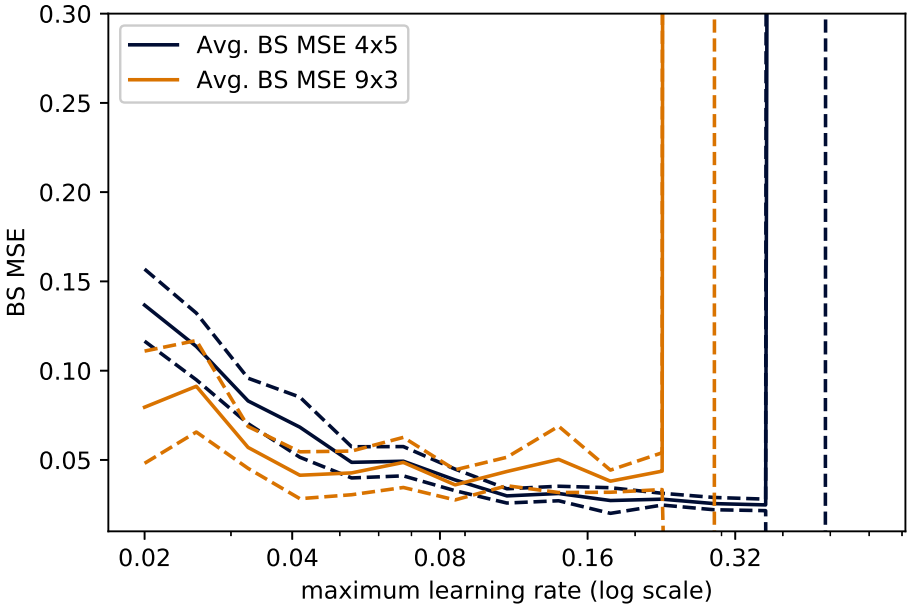


(b)

Figure 3.2: The solid lines represent average sample MSE from 100 models (at each learning rate) where we have removed the worst two out of five models. Dashed lines represent the corresponding 95% condidence bands. The dotted line represent the BS sample MSE, which is the sample MSE of a perfect approximation of the target function. (a) shows average sample MSE for all model designs and (b) shows only 4x5 and 9x3 model design. All models have been trained for to approximate $C^{BS}(s, 100, 0.1, 0.1)$ using the procedure descried in section 2.6 with 30,000 samples, a mini-batch size of 1,024, a minimum learning rate of 10^{-6} and varying maximum learning rate.



(a)



(b)

Figure 3.3: Similar to figure 3.2, but showing average BS MSE over varying maximum learning rates.

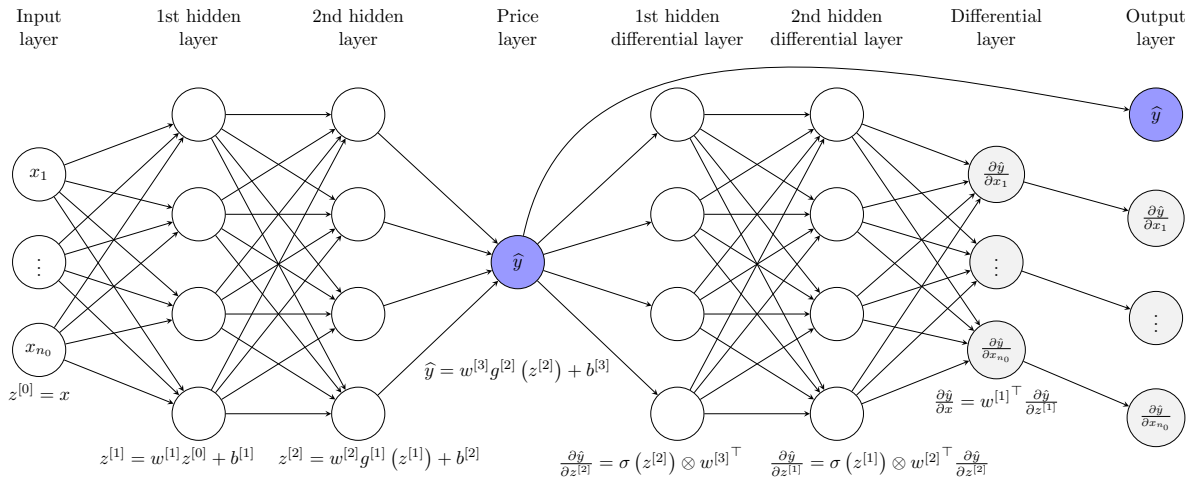


Figure 3.4: Representation of differential neural network

backpropagation algorithm from section 2.3 modified to the output instead of the loss, which means that the extension of the ANN does not introduce any extra parameters.

The objective of the differential neural network is to approximate the target function and its derivative w.r.t. to the inputs, i.e., initial states. It is, therefore, necessary to update the training data to include some sample values that enable the model to learn the derivatives without supplying them directly. We, therefore, update the labels to include the derivatives of the payoffs w.r.t. to the initial states. For estimation of call option prices in the Black-Scholes model, a label would then become the payoff and its derivative w.r.t. the initial state in the sample, i.e.

$$y^{(i)} = \left((S_i(T) - K)^+, \frac{\partial (S_i(T) - K)^+}{\partial S_i(0)} \right)^\top$$

We should note that these labels are not yet normalized. The payoffs are normalized by subtracting the mean and dividing by the standard deviation. This can, however, not be done with the derivative of the payoff, since it will depend on the normalization of the payoff itself. To see this, imagine that our ordinary ANN or prediction rule is represented as a function $f : \mathbb{R} \to \mathbb{R}$, which returns the estimated price in normalized space for some normalized market state. The function, $h : \mathbb{R} \to \mathbb{R}$, that returns the estimated price given a market state in the non-normalized space is then given by

$$h(s) = f\left(\frac{s-a}{b}\right)\beta + \alpha$$

where a, b and α, β are the normalization constants for the market state and payoff, respectively. We can now observe, the derivative of h can be represented as

$$h'(s) = f'\left(\frac{s-a}{b}\right)\frac{\beta}{b}$$

In this setting, the differential ANN can then be represented as a function $f_D : \mathbb{R} \to \mathbb{R}^2$ defined as $f_D(x) = (f(x), f'(x))$. It should now be clear that the derivatives of the payoffs should be normalized by dividing by β/b due to the relation between f' and h'.

We now wish to return to the experiment in the Black-Scholes model. The derivative of the call option payoff w.r.t. the initial state is easily calculated in the Black-Scholes model since $S(T)$ is a Geometric Brownian motion

$$\frac{\partial (S_i(T) - K)^+}{\partial S_i(0)} = \begin{cases} 0 & S_i(T) < K \\ \exp\left(-\frac{\sigma^2}{2}T + \sigma\sqrt{T}Z\right) & S_i(T) > K \end{cases}$$

where Z is a particular realization of a standard normally distributed variable. Note that we ignore the case where $(S_i(T) = K)$ since it is a null event and will not occur in our simulations of $S(0)$. In more general models or with more complicated options, it will most likely not be possible to analytically derive the derivatives of the payoff w.r.t. the initial states. It is, however, always possible to derive derivatives of simulated option payoffs using Automatic Adjoint Differentiation [9], as long as the computation for the payoff consists only (of a possibly large number) of *simple* operations.

We can now create a new weighted loss function

$$\frac{1-\lambda}{m}\sum_{j=1}^{m}\left[f_\theta\left(S_j(0)\right)-\left(S_j(T)-K\right)^+\right]^2+\frac{\lambda}{m}\sum_{j=1}^{m}\left[\frac{\partial f_\theta\left(S_j(0)\right)}{\partial S_j(0)}-\frac{\partial\left(S_j(T)-K\right)^+}{\partial S_j(0)}\right]^2 \tag{10}$$

where λ is the weight for the loss of the derivative. The intention is that the above loss is minimized by the price of the call option and delta in the limit. However it might not be obvious that the derivative of the pricing function actually minimizes the second term of equation (10). We wish to show that this indeed is the case. We consider a more general setting.

Consider a general Markov model with zero interest rate where there exists a function, h, st. the risky asset, $S(t)$ at time t, can be written as a function of the state at time $s < t$ and a multidimensional stochastic process $(X(u))_{u\in(s,t)}$ that is independent of the state at time s

$$S(t) = h\left(s, t, S(s), (X(u))_{u\in(s,t)}\right)$$

This includes the Black-Scholes model, Dupire's local volatility model, Stochastic Volatility models like Heston or SABR, Merton's jump-diffusion model and many more.

Consider now a T-claim, Y, on the risky asset, where we assume that Y can be written as a function of the risky asset from some fixed exposure time T_{EX} (not necessarily 0) to time T. This implies that there exists a function, g, such that the claim is a function of the state at time T_{EX} and of the stochastic process $(X(u))_{u\in(T_{EX},T)}$, ie. $Y = g\left(S(T_{EX}), (X(u))_{u\in(T_{EX},T)}\right)$. The price, f, of T-claim, Y, can at time T_{EX} be represented as

$$f(s) = E\left[g\left(S(T_{EX}), (X(u))_{u\in(T_{EX},T)}\right)|S(T_{EX}) = s\right]$$

Since $(X(u))_{u\in(T_{EX},T)}$ is independent of $S(T_{EX})$ and assuming sufficient integrability of g, the price can be rewritten as an ordinary expectation [8, ex. 4.1.7]

$$f(s) = E\left[g\left(S(T_{EX}), (X(u))_{u\in(T_{EX},T)}\right)|S(T_{EX}) = s\right] = E\left[g(s, (X(u)_{u\in(T_{EX},T)})\right]$$

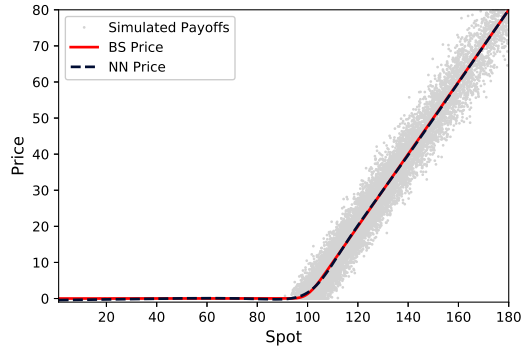
Differentiating w.r.t. s and rewriting the expectation as an conditional expectation (again using [8, ex. 4.1.7]) then yields

$$f'(s) = E\left[g_1(s, (X(u)_{u\in(T_{EX},T)})\right] = E\left[g_1\left(S(T_{EX}), (X(u))_{u\in(T_{EX},T)}\right)|S(T_{EX}) = s\right]$$

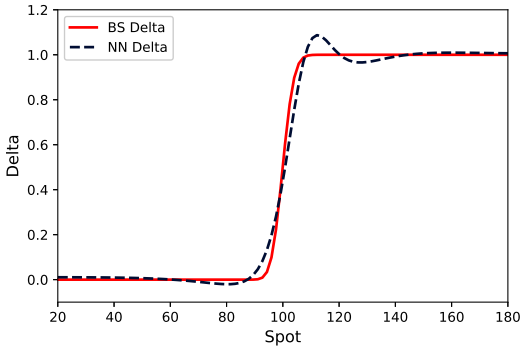
where g_1 denotes the derivative of the function, g, w.r.t. the first entry. This shows that the derivative of the price, f, is just the conditional expectation of the derivative of the payoff. This certainly implies that equation (10) and similar losses for general claims are minimized by the conditional expectation of the claim in the limit. So a differential neural network will (if trained properly) approximate the price and greeks given the initial market state.

The differential neural network cannot be trained using the backpropagation algorithm, as presented in section 2.5, as some parameters/weights are being used more than once in a forward pass. We, therefore, utilize a more general backpropagation algorithm that allows for this kind of recurrence [10]. A forward pass through the differential neural network requires approximately twice as many computations as an ordinary neural network. For this reason, we conduct all comparisons between the ordinary neural network and differential neural network by training the ordinary neural network on 30,000 samples and training the differential neural network both on 15,000 and 30,000 samples.

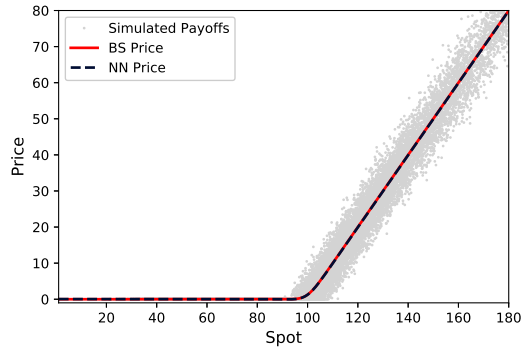
To illustrate performances, we can now conduct the same Black-Scholes fitting experiment as before, only this time utilizing derivatives of the call option payoffs. The result of this test can be seen in figure 3.5. Here we have



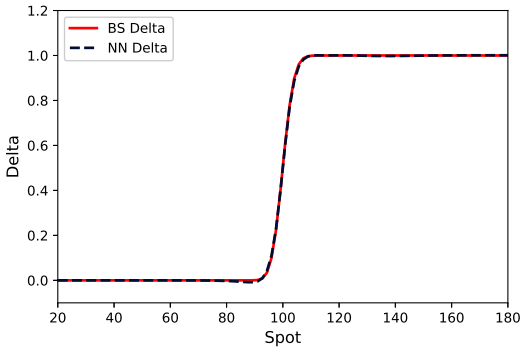
(a) Price from an ordinary 4x5 network trained on 30.000 paths.



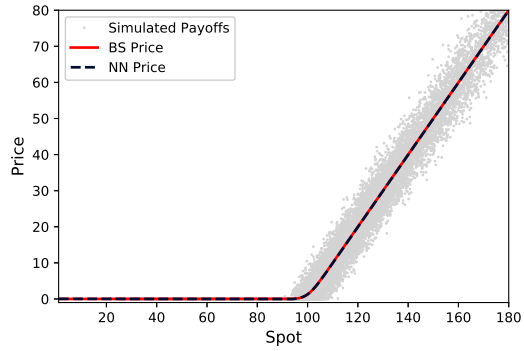
(b) Delta from an ordinary 4x5 network trained on 30.000 paths.



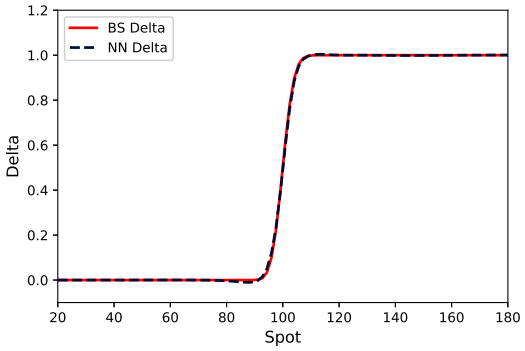
(c) Price from differential neural network with dimensions 4x5 trained on 30.000 paths.



(d) Delta from differential neural network with dimensions 4x5 trained on 30.000 paths.



(e) Price from differential neural network with dimensions 4x5 trained on 15.000 paths.



(f) Delta from differential neural network with dimensions 4x5 trained on 15.000 paths.

Figure 3.5: Fitted call option prices (a)/(c)/(e) and corresponding deltas (b)/(d)/(f) in the Black-Scholes model with $K = 100$, $T = 0.1$ and $\sigma = 0.1$. The ordinary neural network (a)/(b) and the differential neural network (c)/(d) are trained on $m = 30,000$ samples and the differential neural network (e)/(f) is trained on $m = 15,000$ samples. The models have four layers with five units in each and was trained for 100 epoch using mini-batches of size 1024 and a cyclic learning rate with minimum and maximum learning rate of 10^{-6} and 0.1, respectively. In addition, the differential neural networks were trained with $\lambda = 0.9$.

Model design	BS sample MSE	sample MSE	BS MSE	BS Delta MSE
4x5 Ordinary ANN - 30 k	8.2991	8.3133	0.0286	0.001369
4x5 Differential ANN - 30k	8.2991	8.2997	0.0047	0.000009
4x5 Differential ANN - 15k	8.0447	8.0557	0.0036	0.000025

Table 3.2: sample MSE, BS MSE and Delta MSE from ANNs depicted in figure 3.5. BS sample MSE is also included for the two sample sets utilized for training (one with 30,000 samples and one with 15,000 samples).

trained three 4x5 ANNs to approximate $C^{BS}(s, 1, 0.1, 0.1)$ where one is a ordinary feed-forward ANN trained on 30,000 samples (figure 3.5a, 3.5b) and the second is a differential ANN trained on 30,000 samples (figure 3.5c, 3.5d) and the third is a differential ANN trained on 15,000 samples (figure 3.5e, 3.5f), where the differential ANNs are trained with $\lambda = 0.9$. We do not go into detail with the choice of λ, but we have, through simple experimentation, found that setting a dominant weight on the derivative seems superior. The focus of this article will be on the difference between ordinary ANNs and differential ANNs.

It seems clear that both differential ANNs fit the Black-Scholes call option price and its delta very well compared to the ordinary ANN. The error measures for all three ANNs can be seen in table 3.2, including BS sample MSE for the two sample sets (one with 30,000 samples and one with 15,000 samples). Note we cannot compare sample MSE across ANNs trained on different sample sets. However, we do observe that BS sample MSE and sample MSE are relatively close for all ANNs, suggesting that no ANN converged to an unacceptable minimum (or even diverged). The BS MSE is an order of magnitude lower for the differential ANNs when compared to that of the ordinary ANN. The BS Delta MSE is, moreover, several orders magnitude lower for the differential ANNs when compared to that of the ordinary ANN. These performance measures indicate that differential ANNs outperform than ordinary ANNs, even when the differential ANN is only trained on half as many samples.

To truly compare the performance of differential ANNs to ordinary ANNs, we conduct a similar experiment as in the previous subsection: For each model design and each of the 15 maximum learning rates, from 0.02 to 0.6, we train 100 models to obtain 20 performance averages, which are averages of the best three models from groups of five. The results can be seen in figure 3.6 showing BS MSE and BS Delta MSE. We observe that the BS MSE and especially the BS Delta MSE is significantly lower for the differential neural networks when compared to the ordinary ANN. This is also the case for the differential ANN that is only trained on 15,000 samples as opposed to 30,000.

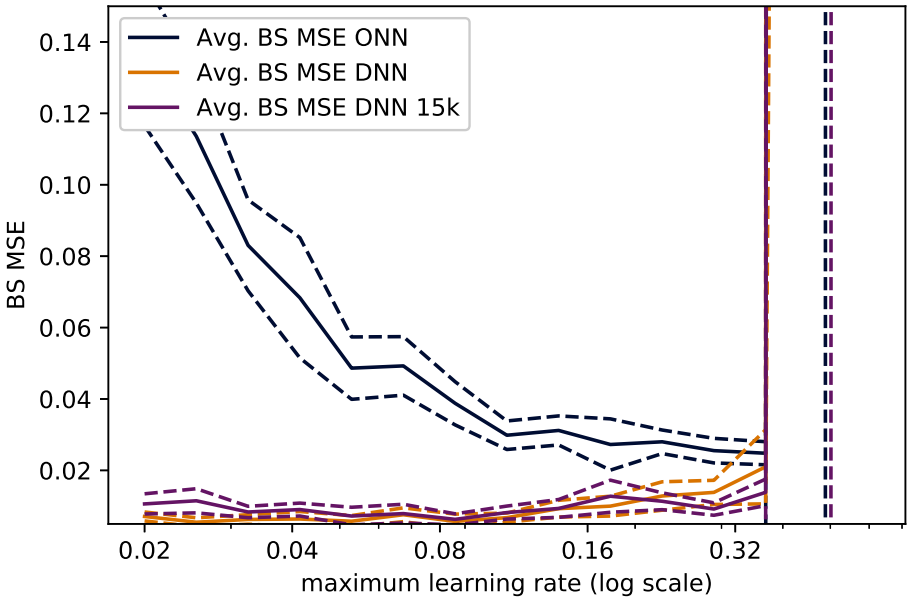
The differential ANN trained on 30,000 samples has a BS Delta MSE of around 0.00002 at its optimal range, which is magnitudes lower than the ordinary neural network. The accuracy of the differential ANNs also seems to be more stable for a wider range of learning rates, which suggests that training on both payoffs and derivatives has a stabilizing and regularizing effect.

This comparison displays the power of differential ANNs. The differential ANNs are still more accurate when trained on half as many samples and they provide a regularizing and stabilizing effect. The added stability is especially useful since training is conducted online, which implies that we do not have much time for retraining or fine-tuning parameters.

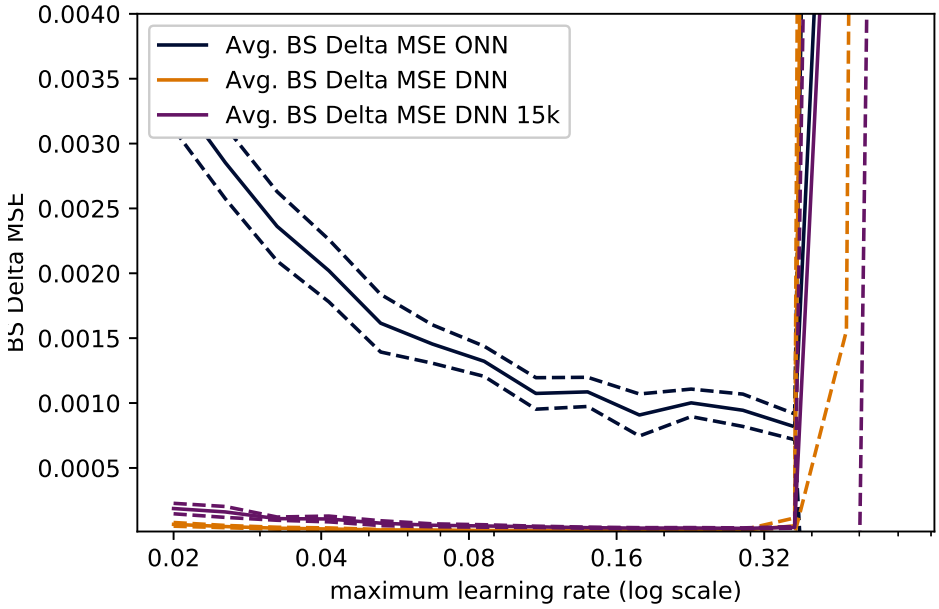
3.4 Comparison of ANNs to Monte Carlo

In this section, we compare the performance of a differential ANN to Monte Carlo simulations. To do this, we consider the performance of the differential ANN on a call option with volatility, $\sigma = 0.2$ and maturity $T = 1$.

All ANNs in this section have four layers with five units in each and are trained for 100 epoch using mini-batches of size 1024 and a cyclic learning rate with minimum and maximum learning rate of $10^{-6}$and 0.1, respectively. In addition, the differential ANNs are trained with $\lambda = 0.9$.



(a)



(b)

Figure 3.6: The solid lines represent average BS MSE (a) and BS Delta MSE (b) from 100 models (at each learning rate), where we have removed the worst two out of five models. Dashed lines represent the corresponding 95% confidence bands. The models all have four layers with five units in each, but one is an ordinary neural network and the two other are differential ANNs. The ordinary and one differential ANNs were trained on 30,000 samples, whereas the last differential ANN was only trained on 15,000 samples. All models have been trained to approximate $C^{\text{BS}}(s, 100, 0.1, 0.1)$ using the procedure described in section 2.6 for 100 epochs with a mini-batch size of 1024, a minimum learning rate of 10^{-6} and varying maximum learning rate.

We fit 1,000 4x5 differential ANNs for different simulated sample sets of 30,000 samples, where 500 sample sets are generated with $\sigma_0 = 0.4$ (as usual) and 500 sample sets are generated with $\sigma_0 = 0.1$. We do this to see the effect of clustering market states in the sample set, in this case, close to strike. We then compute estimated prices for the option given market states, $S(0)$, varying from 60 to 180 in all 1,000 models. This allows us to create two sets of 95% confidence bands for the deviations to the true prices. One set for ANNs trained on sample sets generated with $\sigma_0 = 0.4$ and one set for ANNs trained on sample sets generated with $\sigma_0 = 0.1$.

These confidence bands can be seen in figure 3.7a along with confidence bands for Monte Carlo simulations using 3,000, 6,000 and 30,000 samples. From this figure, it is apparent that the differential ANN is not very reliable on out-of-the-money options when compared to Monte Carlo simulations. This goes for ANNs trained on sample sets generated with both $\sigma_0 = 0.4$ and $\sigma_0 = 0.1$. This is likely due to the low prices of out-of-the-money options. We do, however, see that the ANNs trained on sample set generated with $\sigma_0 = 0.1$ are significantly more accurate around the strike, which is expected since they are trained with more data in this region. Conversely, it is clear that accuracy for the ANNs trained on sample sets generated with $\sigma_0 = 0.1$ diverge quickly away from the strike.

For options with $S(0)$ between 90 and 150, the performance of the 4x5 differential ANN trained on sample sets generated with $\sigma_0 = 0.4$ is like that of a Monte Carlo simulation with 6,000 samples. This does not seem very impressive, but one should be aware of the fact that the differential ANN estimates the price of call options for a whole range of market states or spots, where a Monte Carlo simulation is restricted to a single market state. This shows that it is crucial to consider the range of market states that are necessary to estimate with the ANN.

One could also consider applying various variance reduction techniques. In figure 3.7b, two dimensional Sobol numbers have been utilized to generate the sample data. We will not justify the use of Sobol numbers in this section, but the confidence bands in figure 3.7b show, that the variance of differential ANNs are significantly improved both when the sample sets are generated with $\sigma_0 = 0.4$ and $\sigma_0 = 0.1$. The accuracy of the ANNs is now almost on par with ordinary Monte Carlo simulations for spots between 90 and 120, which is quite impressive. That said, the accuracy still suffers for deep out-of-the-money options.

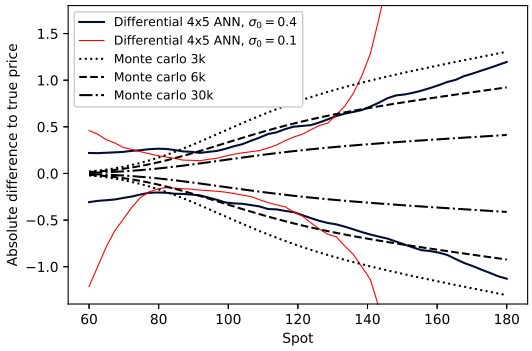
The last variance reducing technique that we wish to apply is averaging ANNs trained on the same sample set. This might not seem like a practical solution due to the added computational load, but one has to remember that in most practical settings, the simulation of the sample set is the most time consuming compared to model training. This is not the case in the Black-Scholes setting with one risky asset, where simulating the sample set is almost instantaneous. We will, nevertheless, apply the technique of averaging ANNs to study its effects.

We train 2,500 4x5 Differential ANNs and average them in groups of five to obtain 500 ANN averages. In each group, the ANNs have been trained on the same sample set generated utilizing Sobol numbers and with $\sigma_0 = 0.4$. In figure 3.7c, we can observe the 95% confidence bands for the ANN averages. It is clear that this technique has a very significant effect on variance: From $S(0)$ between 90 and 180, the performance of the ANN averages are between that of a Monte Carlo simulation with 30,000 and 90,000 paths. The variance for out-of-the-money call options has, furthermore, been reduced significantly, but is still large when compared to Monte Carlo simulations.

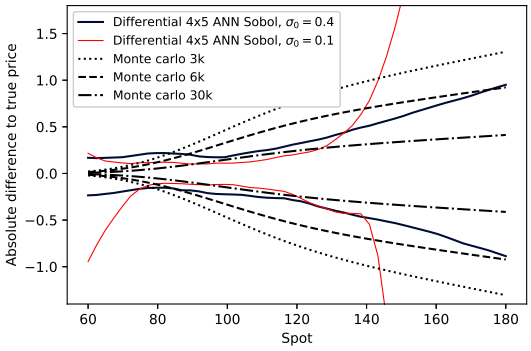
It might be possible to further improve these results by a better choice of model architecture (number of layers and units) or a more appropriate choice of the training procedure. In this article, we do not consider any further improvements.

All in all, this section has shown that ANNs are suited to learn and estimate call option prices in the Black-Scholes model. We obtain the best performance with a moderate number of layers and units and utilization of derivatives of payoffs through differential ANNs drastically improves accuracy. The resulting ANN is not quite on par with Monte Carlo simulations with 30,000 samples, but through the application of Sobol numbers and averaging of ANNs, we are able to close the gap for a large range of market states.

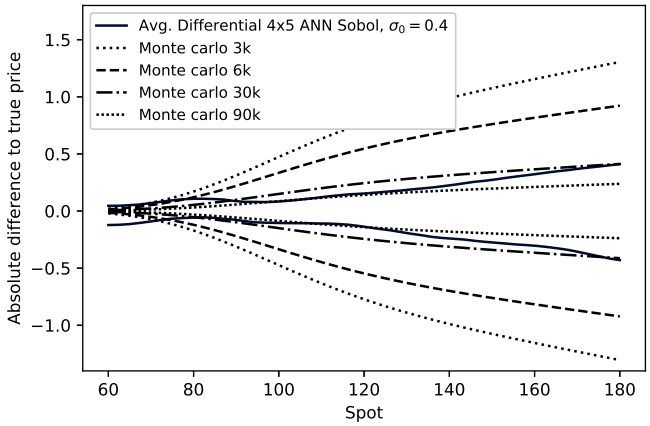
One would still have to consider more exotic options and models to obtain a complete understanding of the capabilities of ANNs for option pricing. See [3] for examples of applications on basket-options and auto-callables.



(a) Samples from random numbers



(b) Samples from Sobol sequence



(c) Average of five ANNs trained on samples from Sobol sequence

Figure 3.7: 95% confidence bands for the deviations to the true price of a Differential ANN and Monte Carlo simulations of 3,000, 6,000 and 30,000 paths (90,000 paths included in (c)). In (a), the samples are generated using simulated random numbers and in (b) and (c), the samples are generated using the Sobol sequence with a random starting point. The differential ANNs represented by a dark blue and red lines train a sample set generated with $\sigma_0 = 0.4$ and $\sigma_0 = 0.1$, respectively. In (c), the confidence bands are based on the average prices of five ANNs trained on the same sample set. The true price is a European call option in the Black-Scholes model with $K = 100$, $\sigma = 0.2$ and $T = 1$.

4 A Volatility Surface Calibration Experiment

In this section, a calibration experiment is conducted. The experiment consists of testing whether or not the ANN is capable of representing the volatility surface of the Heston model and further whether this representation can be utilized in order to calibrate the Heston model to an options market volatility surface.

The fundamental distinction between this approach and the prior approach from section 3 is that, in the former method, simulated option payoffs where used as training data, while this approach will utilize exact implied volatilities instead. The training data is deterministic and the ANN is effectively an interpolation scheme. The motivation for the use of ANNs as a replacement for traditional pricing methods lies in the computational speed of a feedforward pass as opposed to the slow numerical integration schemes traditionally used to calculate prices. This section is largely based on the ideas presented in [1] and [2].

Section 4.1 contains a brief overview of the Heston model and the Anderson-Lake method used to calculate the European call option prices. Section 4.2 contains a detailed overview of the methods used to construct a trained ANN capable of being used to calibrate the Heston model. Section 4.3 contains an analysis of the ANNs performance as a tool for calibration.

4.1 The Heston Stochastic Volatility Model

The volatility surface of the standard Black and Scholes model is a plane in $\mathbb{R}^2$. Thus, the model is insufficient when it comes to fitting a market observed volatility surface. The volatility surface of a stochastic volatility model can attain a variety of shapes and therefore provides a better fit to market data. In this calibration experiment, we consider the Heston model as our default choice.

The Heston model consists of a risk-free asset with deterministic interest rate r and an asset with dividend yield q described by the following dynamics

$$\begin{aligned} dS_t &= (r-q)S_t dt + \sqrt{V_t} S_t dW_1^Q(t), \\ dV_t &= \kappa(\theta - V_t)dt + \sigma\sqrt{V_t} dW_2^Q(t), \end{aligned}$$

where W_1^Q and W_2^Q are Brownian Motions with correlation ρdt under the risk-neutral probability measure. The asset dynamics can equivalently be represented in terms of the forward price,

$$dF_t = \sqrt{V_t} F_t dW_1^Q(t), \qquad F_t := E^Q\left[S_T | \mathcal{F}_t\right]. \tag{11}$$

4.1.1 The Anderson-Lake Method

The popularity of the Heston model is due to the existence of several semi-closed formulas for the price of a European call option [11]. The formula proposed in the original article is numerically unstable and slow. The problems can be remedied by utilizing some of the improvements proposed since the article was released.

In section 4.2.1, the need for a fast and robust way to compute European call option prices in the Heston model is essential. For this purpose, a full implementation of a pricing method, as described in [13], is utilized. This section contains an overview of the results and methods provided in the article.

It will become apparent in section 4.2.1, that it is beneficial to view the price in terms of the forward price of the asset. The characteristic function of the log-forward price is defined as

$$\phi\left(u; v, T, \kappa, \theta, \sigma, \rho\right) = E^Q\left[e^{iuX_T} \middle| V_T = v\right], \qquad X_T = \ln\frac{F_T}{F_0}$$

In the Heston model, a closed-form representation of the function exists. This function is computationally heavy, is prone to cancellation errors, and most importantly has a (countably) infinite amount of singularities. An algorithm for computing this function with a focus on minimizing these issues is provided in [13] in appendix A.

Let α and φ be real constants to be determined later and denote the log-moneyness as $\omega := \ln F/K$. The fundamental challenge in the article is creating a robust integration scheme of the following integral with

$Q(z) := \phi(z-i)/z(z-i)$:

$$I = e^{\alpha\omega} \int_0^\infty \operatorname{Re}\left[e^{-x\tan(\varphi)\omega} e^{ix\omega} Q\left(-i\alpha + x\left(\left(1 + i\tan(\varphi)\right)\right)\left(1 + i\tan(\varphi)\right)\right)\right] dx.$$

The price is then determined through the relationship

$$C(T,F,K) = R(\alpha,F,K) - \frac{F}{\pi} I, \quad R(\alpha,F,K) = F1_{\alpha\leqslant 0} - K1_{\alpha\leqslant -1} - \frac{1}{2}\left(F1_{\alpha=0} - K1_{\alpha=-1}\right).$$

The integrand in I is oscillating, which makes numerical integration particularly challenging. The first step is choosing the constants α and φ, such that this oscillation is minimized. The parameter φ is chosen through the rule

$$\varphi = \begin{cases} \frac{\pi}{12}\operatorname{sgn}(\omega), & r\omega < 0, \\ 0, & r\omega \geqslant 0. \end{cases}, \qquad r = \rho - \frac{\sigma\omega}{v + \kappa\theta T}.$$

Bounds on the optimal α is first determined in part through some simple conditions described on p. 33 in [13] as well as through a significantly more involved root-search problem described in appendix B of the article. The optimal α can then be determined as the solution of the following minimization problem

$$\min_\alpha \ln\phi\left(-i\left(\alpha+1\right)\right) - \ln(\alpha(\alpha+1)) + \alpha\omega.$$

Note that, while the characteristic function is a complex-valued function, evaluating it in $-i(\alpha+1)$ for some α, chosen in accordance with the aforementioned bounds, yields a purely real output.

With the constants determined, the integral is approximated for some step size h through the equations

$$I \approx h\sum_{n=-\infty}^{\infty} w_n f(x_n), \quad x_n = \exp\left(\frac{\pi}{2}\sinh(nh)\right), \quad w_n = \frac{\pi}{2}\cosh(nh)x_n,$$

where f denotes the integrand in I. In practice, the procedure is repeated for a decreasing step size until the difference between two consecutive approximations falls below some threshold.

Despite the fact that the last step is suggested in the article, this project utilized a professionally implemented integration scheme from the Python library Scipy instead. This is due to the slow convergence of the partial sums for particular parameter combinations. The professional implementation is also slightly faster.

The characteristic function is prone to numerical overflow for specific parameter combinations. Due to the rare nature of this issue, parameter combinations that result in such an overflow are silently removed throughout the experiment.

4.2 The 3-Stage Calibration Procedure

In this section, the general structure, as well as optimizations of the calibration procedure of the Heston model, is described. The procedure consists of three parts, with the first two being offline and the last being online.

1. The data generation phase consists of constructing a sample space with implied volatility surfaces as labels used in the neural network training procedure.
2. The training phase consists of training the neural network to represent implied volatility surfaces as a function of state and model parameters.
3. The calibration phase consists of calibrating the model to an observed volatility surface using the trained neural network.

The procedures used in the second stage have already been discussed in section 2.6 and the specific details are presented in section 4.3. Details on the former and ladder phase will now be presented.

4.2.1 Generation of Parameter Space and Volatility Surface

This section outlines the details in the data generation phase of the calibration experiment.

The Heston model consists of more parameters than the Black and Scholes model. A European call option consists of two parameters, strike and maturity. The Heston model consists of seven parameters: spot, initial variance, interest rate, dividend yield, long-term variance, mean-reversion speed and volatility of variance. The higher dimensional function relative to section 3 is harder to approximate for the ANN, but fortunately, it is possible to remove three of the parameters.

The value of the call option is only dependent on the dividend yield in terms of the forward price. Thus by modeling the forward price instead of the spot, the dividend yield becomes obsolete. To see this, first, note that the forward price and spot is identical at maturity. Conditional on the spot being equal to s, the price process of the call option is

$$C(t,s) = e^{-r(T-t)} E^Q \left[(F_T - K)^+ \middle| F_t = s e^{(r-q)(T-t)} \right].$$

Since the above is true for the price process of all models, in particular the Heston model and the Black and Scholes model, the implied volatility is only dependent on the dividend yield through the forward price. Thus the spot and dividend yield are jointly captured through the forward price.

The maturity and strike of the option can furthermore be jointly captured by letting the ANN represent a map of the entire volatility surface instead of a single implied volatility. This does not affect the computational load of the data generation procedure; it does, however, benefit the calibration procedure, since a single feedforward pass will generate the entire surface at once.

Training on the entire surface also has a heuristic benefit. The surface is smooth with clear characteristics, so it seems reasonable to assume that the ANN might benefit from learning the entire surface at once. This idea stems from image recognition, which ANNs are particularly adept at [2].

Thus instead of training the ANN to learn a map from a nine-dimensional parameter space into a single implied volatility, we train the network on the following map with some a priori given parameter space $\Lambda \subset \mathbb{R}^7$ and grid of inverse log-moneyness[4] and maturities,

$$\begin{aligned} &f : \Lambda \to \mathbb{R}^{n_\tau \times n_k} \\ &f\left(F, v, \kappa, \theta, \sigma, \rho, r\right) = \left\{ \sigma^{\text{BS}}\left(\tau_i, k_j\right) \middle| i = 1, \ldots, n_\tau, j = 1, \ldots, n_k \right\}. \end{aligned}$$

Note the distinction between Θ and Λ. The former is the space of all weights in the ANN introduced in section 2.1, while the ladder defines the ranges of parameter values in the Heston model with which the ANN is trained.

The construction of the features (i.e. inputs) can furthermore be optimized. Constructing the parameter space through a uniform grid does not occupy the space in an efficient manner. For this purpose, it is more convenient to use Sobol sequences and consequently transform them through an affine transformation.

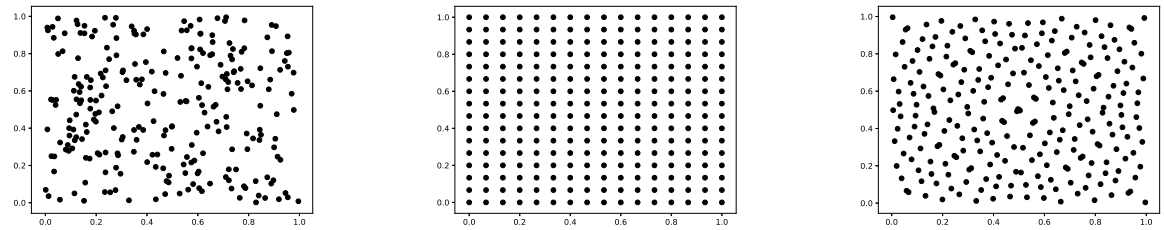
Intuitively, an s-dimensional Sobol sequence is a deterministic sequence of points, x_i, in an s-dimensional unit hypercube I^s such that the error, for any function g, in the following approximation is small [16],

$$\int_{I^s} g(x)dx \approx \frac{1}{N} \sum_{i=1}^{N} g\left(x_i\right).$$

It seems natural that a sequence that adheres to the above criterion must minimize the size of the gaps in the hypercube. An attempt to illustrate this is provided in figure 4.1. It is not entirely clear from the two-dimensional case that the Sobol sequence is superior, but one can hope that this effect is more pronounced in a higher-dimensional space.

Lastly, there is the issue of converting prices to implied volatilities. For some market observed price, p, the implied volatility of the option is the value of the volatility parameter σ, which yields the price of p in the Black and Scholes model. This value is well-defined due to the strictly positive Vega of the Black and Scholes model; it

[4]The inverse log-moneyness is $\ln K/F$.



(a) Uniform sequence (b) Grid sequence (c) Sobol sequence

Figure 4.1: Representation of 256 points of each sequence in the 2-dimensional unit hypercube.

is, however, numerically difficult to determine. The algorithm from [12] is utilized in this project, and the author claims the algorithm capable of producing implied volatilities of arbitrary machine-precision.

To summarize: for some Sobol generated set of parameter combinations, all prices for each combination of maturity and inverse log-moneyness are calculated through the Anderson-Lake method described in section 4.1.1. The prices are then consequently converted to implied volatilities through a publicly available Python implementation of the algorithm developed in [12].

4.2.2 The Calibration Algorithm

This section outlines the details in the model calibration phase of the calibration experiment.

At this stage, it is assumed that some ANN $f_{\widehat{\theta}}$ for $\widehat{\theta} \in \Theta$ is satisfactorily trained with the sample space Λ generated by the procedure outlined in section 4.2.1. The forward price and interest rate parameters are, of course, manually set to the market observable values, so the calibration problem in practice only concerns the five unobservable parameters. For an $n_\tau \times n_k$-dimensional matrix VS representing an observed volatility surface across the same maturity-moneyness-grid as the ANNs codomain, the problem consists of solving the following minimization problem

$$\min_{v,\kappa,\theta,\sigma,\rho} \left\| f_{\widehat{\theta}}(F, v, \kappa, \theta, \sigma, \rho, r) - VS \right\|_F . \tag{12}$$

A problem as the one in equation (12) is naturally solved with a non-linear least squares algorithm. However, such an algorithm is insufficient for this purpose for two reasons.

1. The non-linear least squares algorithm only guarantees a local minimum. The algorithm is, however, often utilized since global optimization algorithms require a large number of function evaluations, which tends to be slow. However, the strength of the trained ANN is the speed of the feedforward pass, so multiple function evaluations are not an issue in terms of speed.

2. As seen in figure 3.1, ANNs can have trouble approximating derivatives, so a gradient-based optimization algorithm might not converge appropriately.

Both issues are remedied by introducing the differential evolution-algorithm. The algorithm is a heuristically motivated gradient-free global optimization algorithm [14]. The algorithm optimizes real functions defined on a multidimensional domain and is therefore capable of solving the problem in (12). In this project, the implementation from the Python library Scipy is utilized.

4.3 Calibration Experiment

The overall procedure for using ANNs in volatility surface calibration is outlined in section 4.2. The natural next step, and the purpose of this section, is to test the procedure in a practical setting.

A test of the 3-stage calibration procedure is two-fold. The quality of fit in terms of the ANNs capability of capturing the shape of the implied volatility surface for the whole parameter space is analyzed in section 4.3.1. The

		F	v	κ	θ	σ	ρ	r	τ	k
Large	Lower bound	75	0.05	1	0.05	0.1	-1	0	0.1	-0.25
	Upper bound	125	0.5	10	0.5	2.0	1	0.1	0.5	0.25
Realistic	Lower bound	75	0.05	1	0.05	0.2	-1	0	0.1	-0.25
	Upper bound	125	0.15	5	0.15	0.5	-0.7	0.1	0.5	0.25

Table 4.1: Parameter bounds used in the data generation stage of the calibration experiment.

quality of fit in terms of the difference between the calibration induced parameters and true parameters is analyzed in section 4.3.2. For the second part, a non-linear least-squares calibration scheme using the Anderson-Lake method is used as a benchmark.

In order to find realistic parameter bounds in the data generation phase, the results from [15] are utilized. The article contains time-series of calibrated parameter values in the Heston model between February 2006 and July 2008. In practice, one would only utilize these bounds, since increasing the boundaries makes for a harder fit, as the ANN must learn from a larger parameter space. However, the bounds obtained from [15] are narrow and all result in an almost linear surface, which is decreasing in log-moneyness. In particular, the correlation is highly negative and volatility low. In the hopes of straining the network further, two sets of parameter spaces are analyzed. A large parameter space producing much more diverse surfaces and a realistic parameter space. The parameter boundaries are summarized in table 4.1.

The performance of the ANN is highly dependent on the size of the sample space. To ensure a proper fit, we generate two hundred thousand data points of the feature space for the large parameter space and one hundred thousand for the realistic.

The corresponding volatility surfaces are calculated on a grid of 8 maturities and 11 inverse log-moneyness values following the methods described in section 4.2.1. The total duration of the data generation stage is around 15 hours for both[5], with the vast majority of time spent on calculating option prices. For both parameter spaces, another ten thousand data points and surfaces used as a test set are generated independently[6] using the exact same settings.

As for the training stage, there is only a need to train a single ANN for each parameter space and this training is conducted offline. This is in sharp contrast to the method analyzed in section 3 where the training is conducted online and as such, one must find a method to disregard degenerate fits and finding learning rates which works best "on average".

The following settings are found to result in a good fit of the artificial neural network. The model is trained for 100 epochs using four hidden layers with 30 units, mini-batches of size 32, and a one-cycle learning rate which dips at 10^{-4} and peaks at 10^{-2}. The learning rates are found through the method described in section 2.6. A graph-representation of the architecture is provided in figure 4.2.

The feature space is normalized with the min-max scaling procedure, such that the minimum and maximum of the data is -1 and 1, respectively. Each combination of maturity and inverse log-moneyness is normalized with the mean-variance scaling, such that each combination has zero mean and unit variance. This normalization entails all the benefits discussed in section 2.6.

In the calibration stage, the test data is used as a proxy for market observed volatility surfaces. That is, one can view the test set as a market observed volatility surface for which the true parameters are known. By using the ANN to calibrate the model on the test set, it is then possible to assess whether or not the calibration found the true

[5]Prices for low volatility and extreme correlation options seem to take longer to calculate, so even though the feature space is cut in half for realistic parameters, the computational time is almost identical.

[6]Independently as in the first two hundred thousand values of the Sobol sequence are skipped before generating the test set.

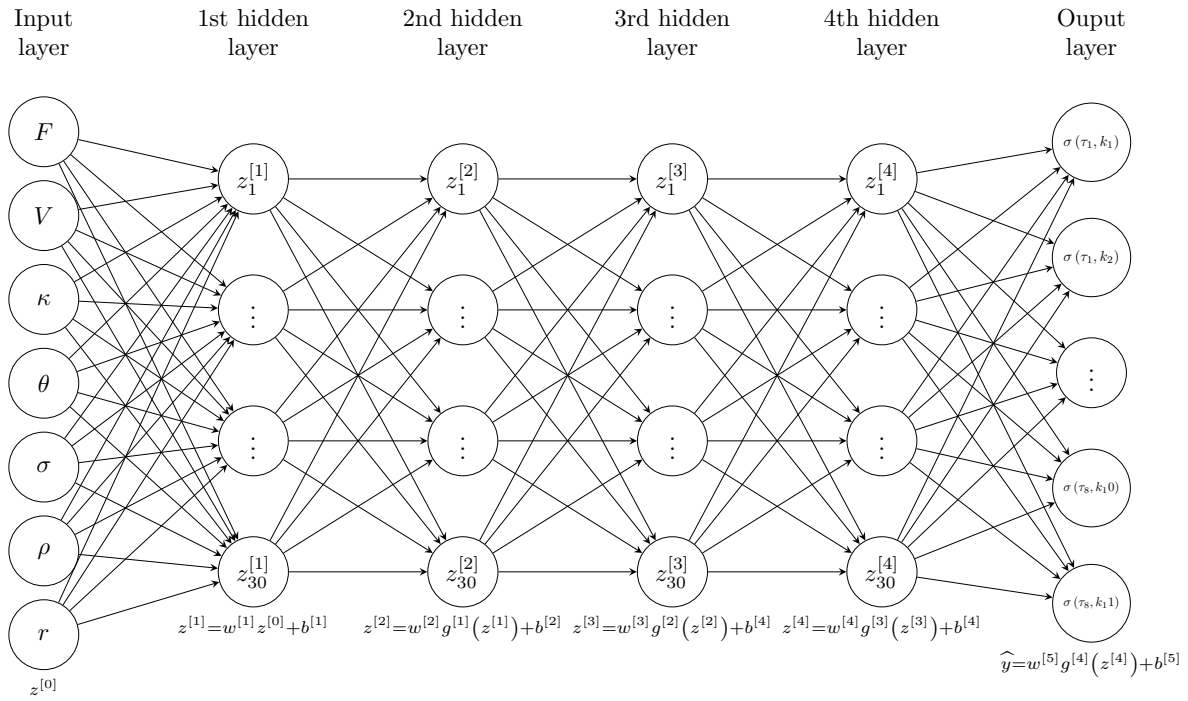


Figure 4.2: The architecture of the ANN used in the calibration experiment.

parameters. If the calibration is reasonably close to the true parameters, it seems reasonable to assume that the calibration procedure can replace the benchmark procedure.

One might claim that the analysis should be based on a market observed volatility surface instead of one generated by the Heston model. This claim is invalid since such an analysis is biased in the sense that the result depends on the Heston model's capability of fitting market observed volatility surfaces. If the fit is insufficient, it is unclear whether the poor performance is due to the ANN or the Heston model. This is discussed in more detail in section 4.3.2.

Since the ANN is trained in a normalized space, it is natural to calibrate in this space as well. This approach simplifies the implementation and ensures that the calibration algorithm does not prioritize any parameters due to different scales.

As a benchmark for performance and speed in calibration, a more classical integration scheme will be used. That is, the Anderson-Lake method will be used to generated volatility surfaces and a non-linear least squares algorithm will conduct the calibration to the test set. The initial guess (in normalized space) for the algorithm is a mean zero and very low variance normal random variable. This test will only be conducted on the first five hundred points of the test set corresponding to the realistic parameter bounds. This is due to the significant increase in the time it takes to perform one calibration with the benchmark.

4.3.1 Quality of Fit in the Model Training Stage

To assess how well the ANN approximates the true implied volatilities, one must use a test set which is generated independently from the training set. As mentioned above, the test set contains ten thousand parameter combinations. The error of each combination of maturity and inverse log-moneyness can be calculated, which is a more informative error measure, than simply calculating the total error. As error measure, the relative error is the natural choice. Formally, for some target value y_{target} and approximation y_{approx}, the relative error is

$$\left| \frac{y_{\text{target}} - y_{\text{approx}}}{y_{\text{target}}} \right|.$$

Note that the errors are not calculated in relation to the normalized labels, even though these are used in training and calibration stages. This is because the scaling results in several labels being very close to zero. Since relative errors require division by these values, the errors are distorted or even undefined.

A heatmap illustrating the mean, standard deviation, and 99% quantile of the relative error between the approximated surfaces generated by the trained ANN and the volatility surfaces in the test set is provided in figure 4.3 for both parameter spaces.

As seen in figure 4.3a, the fit of the large parameter space is worse around the boundary of the surface, especially the corners. The standard deviation of the relative error behaves in much the same manner as the mean, with the highest deviation being located at the corners. The highest standard deviation is around a half percent, which is very high in relation to the average errors.

Finally, we look to the 99% quantile of the error at every point in the surface in figure 4.3e. An error of 1% at the corner is disconcerting, but one has to remember that the standard deviations are small, indicating that these errors are rare. Moreover, the test set is large, so it is expected that the quantile will be as well[7].

The findings indicate that the network performs very well in the middle of the surface, but struggles to approximate the surface perfectly at the edges. The phenomenon is in line with the intuition for using the entire volatility surface at once. The purpose of having the ANN predict entire surfaces was done in the hope that the network would learn the shape of volatility surfaces. Since the network is not provided with information about the shape outside of the boundaries, the network is provided with less information in these areas. An obvious conclusion from this is that one should train the network on an even larger surface and then discard the edges in calibration.

Turning to the more realistic parameter space in figure 4.3b, 4.3d and 4.3f, the errors are almost systematically cut in half relative to the large parameter space (note the different scale of the colour bars). Given the large reduction in the size of the parameter space, the overall improvement in quality is not substantial. An interesting note is that the ANN almost perfectly replicates implied volatility for low inverse log-moneyness. For short time to maturity, low volatility and highly negative correlation, this part of the surface is close to linear, which seems to be easy for the network to replicate.

It is difficult to draw any conclusion from the errors alone. The standard deviations are large relative to the average errors, which might indicate that the calibration will be unstable. On the other hand, the quantile errors are small relative to the average errors, which indicates the exact opposite. This same conclusion holds for the large parameter space.

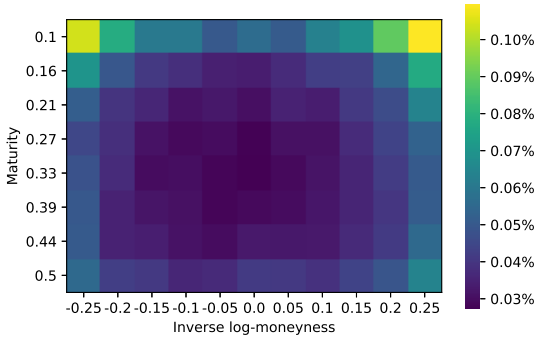
In terms of computational speed, the advantage of the ANN in comparison to the Anderson-Lake method is unprecedented. In the data generation phase, one volatility surface is produced in the span of about 600 milliseconds. The replication of such a surface with a trained ANN runs in the span of 0.34 milliseconds.

4.3.2 Quality of Fit in the Calibration Stage

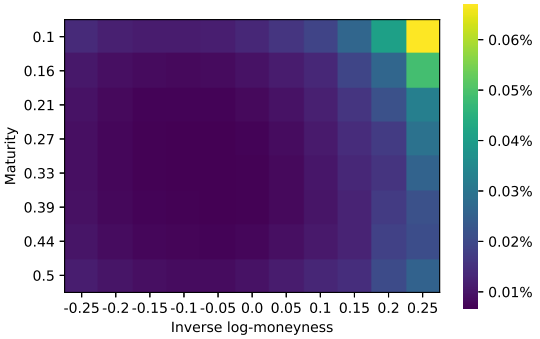
As mentioned above: to asses whether or not one can substitute the numerical integration scheme with a trained ANN in parameter calibration, the test set is again useful. Each row in the test set can be interpreted as an observed volatility surface for which we know the true parameters. One can then compare the true parameters to the one found through calibration. For each row in the test set, the ANN is used to calibrate the model.

In section 4.3.1, it was possible to calculate relative differences in denormalized space, since no implied volatilities were pathologically close to zero. Unfortunately, the parameter ρ ranges from -1 to 1 in both normalized and denormalized space, so using relative differences is impossible. A better measure is to calculate the absolute difference in the normalized space, since this allows for comparison between parameters as well as a decent interpretation. The parameters are scaled between -1 and 1, so the worst possible error is 2. The mean, standard deviation and quantile absolute errors for each parameter are provided in table 4.2 for the benchmark and ANN calibrations.

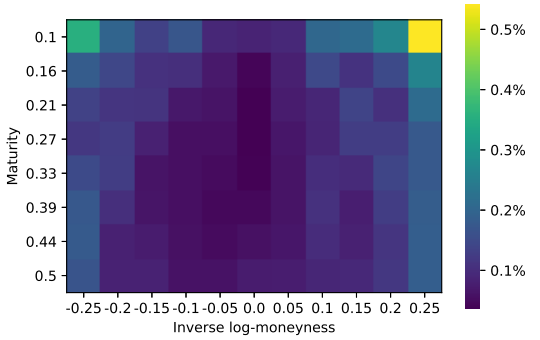
[7]Remember, the quantile error grows weakly as a function of the size of the test set.



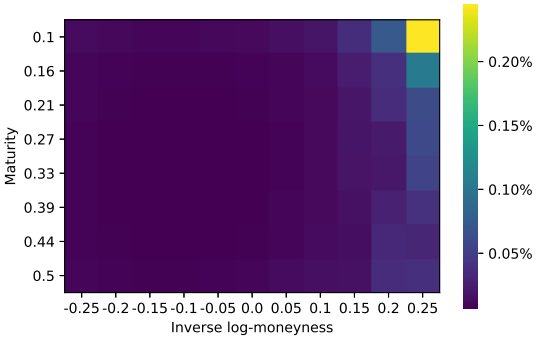
(a) Large Mean Relative Error



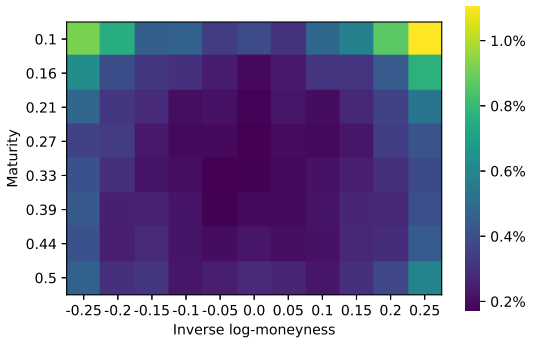
(b) Realistic Mean Relative Error



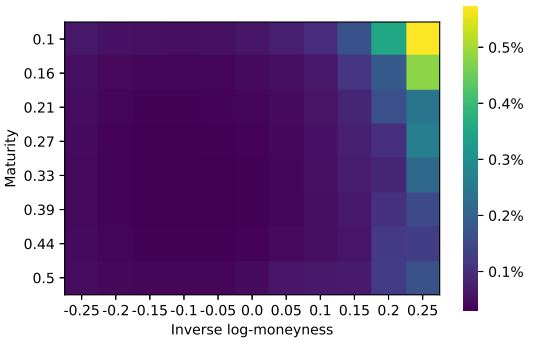
(c) Large Standard Deviation Relative Error



(d) Realistic Standard Deviation Relative Error



(e) Large 99% Quantile of Relative Error



(f) Realistic 99% Quantile of Relative Error

Figure 4.3: Denormalised relative errors of the trained ANN in relation to the test set for the large and realistic parameter space.

		v	κ	θ	σ	ρ	Approximation error	Calibration error
Benchmark	Mean	10^{-10}	10^{-9}	10^{-10}	10^{-9}	10^{-8}	0	10^{-10}
	Standard Deviation	10^{-9}	10^{-8}	10^{-9}	10^{-8}	10^{-7}	0	10^{-10}
	99% Quantile	10^{-9}	10^{-8}	10^{-8}	10^{-7}	10^{-7}	0	10^{-9}
Large	Mean	0.002	0.014	0.005	0.013	0.013	0.0023	0.0016
	Standard Deviation	0.002	0.016	0.011	0.017	0.035	0.0022	0.0013
	99% Quantile	0.007	0.075	0.048	0.078	0.178	0.0116	0.0054
Realistic	Mean	0.001	0.005	0.002	0.005	0.013	0.0005	0.0004
	Standard Deviation	0.001	0.005	0.004	0.005	0.018	0.0004	0.0002
	99% Quantile	0.003	0.023	0.017	0.024	0.079	0.0019	0.0012

Table 4.2: Descriptive statistics concerning the absolute calibration errors for each parameter in the calibration experiment as well as approximation errors and calibration errors.

There is an important distinction to be made between the three sources of error in a market calibration. To aid in the discussion, refer to the Frobenius norm of the difference between two surfaces as the surface error[8]. The three sources of error are as follows:

1. The **approximation error** concerns the inherent difference between the analytical volatility surface and the volatility surface created by some approximating function, e.g., an ANN. The analytical volatility surface refers to the volatility surface of the Heston model if it could be calculated without some kind of numerical error. For the sake of discussion, we will, throughout this section, consider the surface produced by the Anderson-Lake method as the analytical surface. So the approximation for some $\lambda \in \Lambda$ and approximating function f is
$$\sqrt{\sum_{i=1}^{n_\tau}\sum_{j=1}^{n_k}\left[f\left(\lambda,\tau_i,k_j\right)-\sigma^{\mathrm{BS}}\left(\lambda,\tau_i,k_j\right)\right]^2},$$
where σ^{BS} refers to the numerical estimation of the implied volatility using the Anderson-Lake method. Under this assumption, the approximation error is zero for the benchmark method. The approximation errors for the ANNs is given in table 4.2. The approximation error is only important in the sense that it influences the calibration error.

2. The **calibration error** concerns the surface error between the analytical surface induced by respectively the calibrated parameters and the true parameters. That is, if $\lambda, \widehat{\lambda} \in \Lambda$ represent the true and calibrated parameters and σ^{BS} represents the analytical surface of the Heston model, then the calibration error is
$$\sqrt{\sum_{i=1}^{n_\tau}\sum_{j=1}^{n_k}\left[\sigma^{\mathrm{BS}}\left(\lambda,\tau_i,k_j\right)-\sigma^{\mathrm{BS}}\left(\widehat{\lambda},\tau_i,k_j\right)\right]^2}.$$
In practice σ^{BS} is calculated using the Anderson-Lake method. This error is presented in table 4.2 for the ANN-calibrated parameters and the benchmark calibrated parameters. This error allows us to compare the performance of different calibration procedures.

3. The **model error** concerns the failure of the Heston model to accurately represent a market observed volatility surface. This error can be measured as the surface error between the analytical surface induced by the calibrated parameters and the market observed surface. In this section, analytical surfaces of the Heston model is used as market observed volatility surfaces in the test set, so by construction of the experiment the model error is zero.

Consider first the results of the benchmark calibration. The method rediscovers the true parameters almost perfectly. This might seem like an obvious result, but one has to remember that the gradient is numerically estimated in the

[8]See equation (12) for an example of a surface error.

non-linear least squares algorithm. It could have been the case that the pricing algorithm is too unstable to estimate gradients numerically. The results show, that the benchmark calibration procedure is stable.

The calibration error is unsurprisingly very small. This is a result of the fact that the benchmark method has an approximation error of zero, such that the objective function of the benchmark method is to minimize the calibration error.

As for the ANN trained on the large parameter space, the mean errors indicate that the calibration scheme, on average, rediscovers the true parameters up to the second decimal (in normalized space). One might have suspected a large standard deviation in errors, due to the large standard deviation in figure 4.3. The standard deviations are, however, more favorable in the calibration and they suggest that the vast majority of calibration attempts are correct up to the second decimal (in normalized space). The quantile error for σ and ρ does, however, suggest that the result of some calibrations deviate significantly from the true parameters.

Since the ANN method, in contrast to the benchmark method, has a strictly positive approximation error, it is unable to minimize the calibration error directly. This is apparent in table 4.2, where the calibration error is of the order of magnitude of 10^{-3}.

As for the ANN trained on the realistic parameter space, the calibration rediscovers the true parameters accurately up to the third decimal on average. The standard deviations are significantly reduced, which is probably a result of the significant decrease in the quantile errors. The calibration error is also one order of magnitude smaller than the large parameter space, which is 10^{-4}.

The reduced parameter space results in an improved calibration for two reasons. First, the approximation errors are superior as first seen in section 4.3.1 and then in table 4.2. Moreover, the search space of the global optimum is significantly reduced, which, of course, allows for more precise calibration.

The original motivation for using ANNs in a calibration setting was to utilize the uptick in the computational speed of a forward pass. Compared to the benchmark, the use of ANNs provides a significant decrease in computational time. A benchmark calibration terminates on average after 87 seconds (on a modest CPU). In contrast, the calibration using ANNs terminates within three seconds on average. Furthermore, the result is a global minimum, as opposed to the benchmark.

From the experiment, the conclusion seems quite clear: if one can accept an accurate calibration up to the third decimal, then the ANN-approach is the far superior choice in terms of computational speed. However, this experiment ignores two important aspects which are relevant if the test set consisted of market observed volatility surfaces and not the analytical surfaces.

The first aspect is the fact that the calibration procedure presented in this section, as opposed to traditional calibration schemes, necessitates a definition of the grid of points with which the ANN can calculate implied volatilities. This restriction entails that an observed volatility surface must be defined on the same exact grid points before calibration can commence. The only way to ensure this is through some pre-processing step where the observed surface is interpolated.

The second aspect concerns the fact that this analysis does not reveal anything about the model error since there is not used any market observed volatility surfaces. In [2, figure 12], the historical surface error between a calibrated rough volatility model's surface and the SPX induced volatility surface between 2010 and 2019 is presented. The surface errors range between 0.0025 and 0.02. The calibration procedure is similar to the benchmark method, but the values are, of course, still not directly comparable to the Heston model. It does, however, seem reasonable to assume that a similar analysis of the Heston model would yield surface errors of the same order of magnitude (or worse). These errors can, therefore, be used as a proxy for the model error in the Heston model.

A possible interpretation of the errors is as follows: it does not matter that the calibration error of the ANN-methods is larger than the benchmark since the model error is larger than both calibration errors. So the extra accuracy obtained by using the benchmark method as opposed to using ANNs is not worth the increase in computational time.

The results show that there are strong arguments in favor of and against using ANNs as a means to calibrate the Heston model. While the significant increase in computational speed cannot be ignored, the ANN calibration scheme has a much larger calibration error. The calibration error for the realistic parameter space is, however, still one order of magnitude smaller than the model error, so one could argue that this deviation is insignificant.

5 Conclusion

In this project, artificial neural networks and their applicability in a financial setting is analyzed from a theoretical and practical point of view.

In section 2, artificial neural networks are presented and the theoretical basis for their use is established. The universal representation theorem is presented, which shows that shallow artificial neural networks are capable of approximating continuous functions under mild restrictions on the activation function. A strong argument in favor of deep artificial neural networks is further proven. The backpropagation algorithm is derived in a general setting with an arbitrary number of samples.

Strong arguments in favor of depth in a financial setting are further given in section 3. Through an introduction of the concept of approximating option pricing functions by training artificial neural networks on simulated option payoffs in the Black and Scholes model, the benefit of depth is analyzed. The results show that the best performance is achieved through a modest balance between depth and width.

Section 3 further shows how the approximation, in particular of the option delta, can be significantly improved by utilizing differential ANNs, where the loss function includes derivatives of option payoffs. The differential ANNs also exhibit more stability to the applied learning rate, which increases stability.

Lastly, section 3 shows that performances of ANNs for option pricing, when compared to ordinary Monte Carlo, depends strongly on the considered range of market states used for training of the ANN. The resulting ANNs are inferior without the application of variance reduction techniques. Utilizing Sobol numbers and averaging ANNs is, however, enough to close this gap for a large range of market states.

The focus of section 4 is the use of artificial neural networks as a tool for fast model calibration. By using the network to interpolate generated volatility surfaces from the Heston model, the computational speed of generating volatility surfaces is improved by several orders of magnitude. It is shown that this interpolation is not as precise as the benchmark scheme, but does yield small enough errors to be considered as a potential replacement. It is also clear from the analysis that satisfactory performance is only achieved by making strong, but realistic assumptions about the range of possible parameter values.

To summarize: this article shows how artificial neural networks can be utilized in option pricing and model calibration. While promising results were obtained in both areas, it also became apparent that the theory of artificial neural networks requires a significant amount of work to apply in practice.

References

[1] McGhee, W. A. (2018). *An artificial neural network representation of the SABR stochastic volatility model.* Available at SSRN 3288882.

[2] Horvath, B., Muguruza, A., & Tomas, M. (2019). *Deep learning volatility.* Available at SSRN 3322085.

[3] Huge, B., & Savine, A. (n.d.). *Deep Analytics.* Retrieved from https://www.deep-analytics.org/antoine-savine-brian-huge-deep-analytics

[4] Björk, T. (2009). *Arbitrage theory in continuous time.* Oxford university press.

[5] Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization.* arXiv preprint arXiv:1412.6980.

[6] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT press.

[7] Telgarsky, M. (2015). *Representation benefits of deep feedforward networks.* arXiv preprint arXiv:1509.08101

[8] Durrett, R. (2019). *Probability: theory and examples* (Vol. 49). Cambridge university press.

[9] Savine, A. (2018). *Modern Computational Finance: AAD and Parallel Simulations.* John Wiley & Sons.

[10] Savine, A. *Computational Finance Lecture Slides.* Retrieved from https://github.com/asavine/CompFinLecture/blob/master/lectureSlides.pdf

[11] Heston, S. L. (1993). *A closed-form solution for options with stochastic volatility with applications to bond and currency options.* The review of financial studies, 6(2), 327-343.

[12] Jäckel, P. (2015). *Let's be rational*. Wilmott, 2015(75), 40-53.

[13] Andersen, L. B., & Lake, M. (2018). *Robust High-Precision Option Pricing by Fourier Transforms: Contour Deformations and Double-Exponential Quadrature.* Available at SSRN 3231626.

[14] Storn, R., & Price, K. (1997). *Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces.* Journal of global optimization, 11(4), 341-359.

[15] Guillaume, F., & Schoutens, W. (2010). Use a reduced Heston or reduce the use of Heston?. Wilmott Journal, 2(4), 171-192.

[16] Sobol', I. Y. M. (1967). *On the distribution of points in a cube and the approximate evaluation of integrals.* Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki, 7(4), 784-802.

[17] Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function.* Neural networks, 6(6), 861-867.

[18] LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K. R. (2012). *Efficient backprop.* In Neural networks: Tricks of the trade (pp. 9-48). Springer, Berlin, Heidelberg.

[19] Smith, L. N. (2017, March). *Cyclical learning rates for training neural networks.* In 2017 IEEE Winter Conference on Applications of Computer Vision (WACV) (pp. 464-472). IEEE.

[20] Schilling, R. L. (2017). *Measures, integrals and martingales.* Cambridge University Press.