Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Estimating scenario-based Initial Margin for financial portfolios using Neural Networks and Random Forests

ALEKS PETKOV

Estimating scenario-based Initial Margin for financial portfolios using Neural Networks and Random Forests

ALEKS PETKOV

Master in Computer Science
Date: March 2, 2025
Supervisor: Carlos Barreto Suarez
Examiner: Mats Nordahl
School of Electrical Engineering and Computer Science
Host company: OSTTRA
Swedish title: Uppskattning av scenariobaserad Initial Margin för finansiella portföljer med Neurala Nätverk och Random Forests

Abstract

Financial institutions, such as banks and central counterparties (CCPs), often need to calculate a scalar quantity called initial margin (IM) to manage credit risk and cover potential losses from various transactions. IM is typically calculated using mathematical models such as SIMM or scenario-based CCP IM models. However, CCP IM models are computationally expensive because they require repeatedly considering a large number of observed market movements when only a few of said movements are actually relevant for calculating the final IM value. Finding an alternative way to calculate IM which eliminates these redundant calculations would allow CCPs to more efficiently calculate IM and thereby help market participants reduce their credit risk.

This project utilizes machine learning to train feedforward neural networks to estimate IM for financial portfolios. A random forest model is also trained to provide a point of comparison for the performance of the neural networks. The two datasets used in this study are generated using portfolio data from 40 anonymous banks and their respective IM labels are obtained using SIMM and a proprietary CCP IM model.

The final optimized neural nets have strong performances and estimate IM with mean absolute percentage errors of 1.50% and 0.49% on unseen data of the CCP IM and SIMM datasets, respectively. The final random forest models perform worse with errors of 12.3% and 17.3% on the respective datasets. The results suggest that neural networks are well suited for calculating portfolio IM and that they are more efficient than SIMM and the CCP IM model used in this study. However, due to limitations in the dataset generation, more research needs to be conducted to assert that these results are generalizable to more realistic portfolio data.

Sammanfattning

Finansinstitut som banker och clearinghus (även kallade CCP:er) behöver ofta beräkna en skalär storhet som kallas initial margin (IM) för att hantera kreditrisk och täcka potentiella förluster vid olika transaktioner. IM beräknas vanligtvis med matematiska modeller som SIMM och scenariobaserade CCP IM-modeller. Dessa CCP IM-modeller är däremot beräkningsmässigt ineffektiva då de kräver upprepad behandling av en stor mängd olika marknadsrörelser när endast en liten andel av dessa bidrar till det slutgiltiga IM-värdet. Att hitta en alternativ beräkningsmetod för IM som kan eliminera dessa beräkningar skulle därmed öka effektiviteten av clearinghus och bidra till att minska marknadsaktörers kreditrisk.

Detta projekt använder neurala nätverk för att träna feedforward neurala nätverk för att uppskatta IM för finansiella portföljer. Som referenspunkt tränas även en random forest-modell vars prestanda kan jämföras mot prestandan av de neurala nätverken. Denna studie genererade två dataset från portföljdata från 40 anonyma banker och portföljernas motsvarande IM-värden erhölls med både en CCP IM-modell och med SIMM.

De slutgiltiga optimerade neurala nätverken presterar utmärkt och uppskattar IM med ett absolut medelfel på 1.5% och 0.49% av de förväntande IM-testvärdena beräknade med CCP IM-modellen respektive med SIMM. De optimerade random forest-modellerna hade däremot sämre prestanda med motsvarande medelfel på 12.3% respektive 17.3%. Resultaten visar på att neurala nätverk är vällämpade för beräkning av scenariobaserad IM och beräkningsmässigt mer effektiva än konventionella IM-modeller. Däremot bygger resultaten på portföljdata som på grund av studiens begränsade dataunderlag inte är fullständigt realistisk, och därmed skulle ytterligare forskning krävas för att säkerställa att resultaten kan generaliseras till mer realistisk portföljdata.

Acknowledgements

I would like to thank my supervisor and my examiner at KTH, Carlos Barreto Suarez and Mats Nordahl, for their guidance throughout the project and for their leniency towards my frequently late submissions and drafts. I also want to thank my wonderful supervisor Ariyan Abdulla at OSTTRA, who helped me each step of the way. Finally, I want to extend my most heartfelt gratitude to my brother for cooking me delicious dinners every day, for too many consecutive weeks, while I was toiling away to finish this report.

Aleks Petkov,
January, 2025

Contents

Acronyms

AdaGrad Adaptive Gradient Algorithm

Adam Adaptive Moment Estimation

ANN Artificial Neural Network

CCP Central Counterparty

DV01 Dollar Value of a Percentage Point

ES Expected Shortfall

FNN Feedforward Neural Network

FX Foreign Exchange

GD Gradient Descent

GS Grid Search

HPO Hyperparameter Optimization

IM Initial Margin

MAPE Mean Absolute Percentage Error

ML Machine Learning

MLP Multilayer Perceptron

PnL Profit and Loss

ReLU Rectified Linear Unit

RF Random Forest

RS Random Search

SGD Stochastic Gradient Descent

SIMM Standard Initial Margin Model

SLP Single-layer Perceptron

VaR Value at Risk

Chapter 1

Introduction

With machine learning (ML) gaining prominence across various scientific fields during the past decade, it has seen widespread use in countless data heavy application domains such as health care, marketing, education and finance [1, 2]. The use of artificial neural networks (ANNs) has especially increased in the finance sector where they have been effective at stock market predictions, credit scoring, loan evaluation, and at predicting bankruptcies in banks and firms [3].

Many studies indicate that neural networks often outperform other traditional statistical techniques, such as discriminant and regression analysis, due to their ability to capture complex and nonlinear patterns in the dataset [4]. This thesis will cover a financial topic which has not yet been widely researched - namely, the use of ML for calculating scenario-based initial margin (IM) in the financial derivatives market.

Section 2.1 explains what IM is and what algorithms are currently used in practice for calculating it. In short, IM is an important risk measure for financial institutions and it is typically calculated using either public mathematical models such as the Standard Initial Margin Model (SIMM) or internal, so-called CCP IM models that are used by central counterparties [5]. A problem that can be identified with the latter models is that they perform a significant amount of unnecessary computations because they require taking into account a large number of observed market movements (so-called *scenarios*), when only a small percentage of said scenarios are actually needed to obtain the final IM value.

The goal of this project, conducted at and with the help of the company OSTTRA, is to utilize ANNs and the bank portfolio data provided by OSTTRA to train feedforward neural networks (FNNs) to approximate two math-

ematical IM models (described further in 2.1.3 and 2.1.4). To provide a point of comparison for the performance of the FNNs, random forest (RF) models will also be trained and evaluated on the IM models.

1.1 Research questions

This study aims to answer the following questions:

1. Can a feedforward neural network (FNN) be used in practice to estimate IM calculated using either a scenario-based CCP IM model or the SIMM model?

2. Are random forest (RF) models able to achieve comparable performance to FNNs when estimating either of the above-mentioned IM models?

3. Are FNNs able to compute scenario-based IM more efficiently than conventional IM models such as SIMM and the CCP IM model available to this project? If so, by how much?

1.2 Hypothesis

The respective hypotheses for the research questions stated above are:

1. An optimized FNN will be able to estimate both SIMM and CCP IM for new portfolios with an average estimation error of less than 10% of the expected IM value. Additionally, the FNN will have a smaller estimation error on the SIMM model.

2. The optimized Random Forest models will have larger estimation error than their FNN counterparts, possibly with error margins larger than 10%.

3. The FNNs will be significantly more efficient at calculating IM than the CCP IM model and SIMM.

1.3 Scope

The scope of this project is limited to:

1. processing the portfolio data that is provided by OSTTRA and generating new portfolios to create sufficiently large and diverse datasets

2. implementing and optimizing FNNs to estimate two different models for calculating IM for portfolios, and comparing the performance of the FNNs to that of RFs.

3. Comparing the computation times of calculating IM using FNNs compared to using the CCP IM model and SIMM.

1.4 Outline

This paper is structured as follows. Section 2 gives an extensive background on and understanding of ANNs, introduces necessary financial concepts and the IM models relevant to this study, and describes previous related research. Section 3 describes the provided portfolio data, the dataset generation and the chosen model evaluation strategy. Section 4 describes the model training and hyperparameter optimization process, and presents the accuracy and speed of the best-performing FNN models. Section 5 discusses the obtained results, their generalizability with respect to the limitations of the study, and compares them to the results of previous work. Section 6 summarizes the conclusions that can be drawn from the study and provides suggestions for future work.

Chapter 2

Background

This chapter aims to provide a domain-specific introduction to financial instruments and explain relevant terminology, as well as to give the reader a thorough, technical understanding of the chosen ML models. It also describes cross-validation, hyperparameter optimization algorithms and some relevant hyperparameters for each model. Finally, this chapter discusses previous research on the topic of using both ML and non-ML models for IM calculations and relates it to this study.

2.1 Finance and terminology

Section 2.1.1 gives a general overview of financial instruments, while Section 2.1.2 explains important concepts and terminology which provides essential context for understanding this study.

2.1.1 Financial instruments

For the purposes of this study, a financial instrument - or simply *an instrument* - can be defined as:

> a real or virtual document representing a legal agreement involving any kind of monetary value [6].

(Note that this is a simplified version of the exact definition given by the *International Accounting Standards* [7].) There are three different types of instruments: cash instruments, derivative instruments and foreign exchange instruments. A cash instrument is one whose value is directly determined by the market - stocks, loans and deposits are some common examples. A derivative

instrument, on the other hand, is one whose value is indirectly determined by underlying risk factors such as assets or interest rates. Examples of derivative instruments are forwards, options, and swaps.

This project deals with the final type of financial instrument - namely, the foreign exchange (FX) instrument. Foreign exchange is the process of exchanging one national currency for another [8]. An FX instrument is thus said to operate on a given *currency pair* - the currency pair USD/EUR would signify a trade involving the U.S. dollar and the euro.

The most common FX instrument is the spot trade, a simple bilateral transaction where two parties immediately exchange a specified amount of currency for a specified amount of another currency. The exchange rate between the currencies in a spot trade is called the spot rate [9]. Spot trades can be contrasted with forward trades, where the exchange rate (now called the forward rate) is agreed upon in advance, but the transaction is made at a specified later date [10].

There are also other, more complex FX instruments such as currency swaps and options whose value also might depend on the forward rates for multiple future dates, such as after one week, one month, one year, etc.

2.1.2 Relevant concepts and terminology

Pricing and sensitivities

The value of an instrument is determined by a pricing function V unique to the type of the instrument, with V being a function of one or more risk factors x_i

$$V = V(x_1, x_2, \ \ldots, x_n).$$

As a simplified example, the value of a one-year forward contract $V(x_1, x_2)$ for exchanging USD into SEK would depend on the spot rate x_1 and the one-year forward rate x_2 for the USD/SEK currency pair. As these exchange rates fluctuate over time, so does the value of the instrument - these fluctuations are known as an instrument's *sensitivities* to those risk factors and they are given by how much the instrument value changes from a 1% increase in each individual risk factor x_1 and x_2:

$$s_1 = V(1.01x_1, x_2) - V(x_1, x_2) = V_1(1.01x_1) - V_1(x_1)$$
$$s_2 = V(x_1, 1.01x_2) - V(x_1, x_2) = V_2(1.01x_2) - V_2(x_2).$$

These sensitivities are also often called DV01 values, which stands for ‘the dollar value of a percentage point’. Each DV01 value thus represents how much the instrument’s dollar value would change if the corresponding exchange rate were to increase by 1% [11].

Portfolios

A financial portfolio is a collection of instruments and its value is simply the aggregated value of all its constituent instruments. Similarly, its sensitivity against a risk factor is the sum of the DV01 values of its constituent instruments against that risk factor.

In this project, a portfolio is comprised of a set of FX instruments covering up to 26 unique currency pairs, each with a corresponding spot rate and 10 forward rates covering a time span of between one week and five years in the future. In total, a portfolio thus consists of 286 different DV01 values - one for each spot and forward rate of each currency pair. These DV01 values are given as input data and are not computed using pricing functions as part of this study.

Market risk

Market risk is the risk of financial losses for a portfolio incurred by market movements that affect the risk factors in that portfolio.

Counterparty credit risk

Trades and transactions between two parties (called bilateral trades) also involve a counterparty credit risk, or simply credit risk, for both parties - it is the risk incurred by the possibility that the ‘other’ party, the counterparty, defaults and does not meet its financial obligations.

Scenarios

A scenario is a set of observed market movements represented as a change to each of the exchange rates in a portfolio. The impact a scenario has on the value of a portfolio is thus given by the sum of multiplying each exchange rate’s DV01 value with the corresponding exchange rate’s observed change.

Initial margin

To mitigate the counterparty credit risk of a trade, both parties involved are required to post a collateral payment to each other, a so-called initial margin (IM), in the form of cash or liquid assets [5]. These IM payments can be thought of as an insurance paid by both parties which would cover potential losses for the counterparty in the event that they default. The initial margin requirements are formally defined by the Basel Commitee on Banking Supervision (BCBS) and the International Organization of Securities Commissions (IOSCO) so as to cover potential future exposures that could be caused by "extreme but plausible" fluctuations in the value of an instrument over a 10-day period, based on historical data during periods of financial stress [12].

Initial margin optimization

While posting IM serves to stabilize the financial market and ameliorate the consequences of market crises that can occur when large market players default, it nonetheless presents a large alternative cost for said players who are unable to use the capital they must set aside for posting IM. It is therefore desirable for market players to adjust their portfolios so as to reduce their counterparty credit risk which would decrease the IM they post and thus free up that capital. This can be formulated as an optimization problem where the objective function to be minimized is the IM of a portfolio and the variables are the instruments that comprise the portfolio. By adding specific instruments to the portfolio, it is possible to reduce the portfolio's credit risk without affecting its market risk (and consequently, its profitability). Doing so, however, requires simulating future IM (also known as *dynamic* IM) for multiple points in time as well as for multiple scenarios, which typically result in nested simulations with a prohibitively high computational cost using models such as SIMM or other scenario-based quantile models (as described in 2.1.3 and 2.1.4, respectively) [13, 14].

Central counterparties

To avoid taking on credit risk, parties can *clear* their trades through a central counterparty (CCP). CCPs are highly regulated financial institutions that provide services to manage credit risk by replacing bilateral trades with two identical contracts between the CCP and each of the counterparties involved in the trade [15]. Instead of posting collateral IM payments to each other, the parties involved in the trade now post that IM to the CCP, and by doing this, the

CCP has effectively taken on the credit risk that the counterparties themselves would otherwise be exposed to.

2.1.3 Standard Initial Margin Model

The Standard Initial Margin Model (SIMM) is a mathematical model for calculating IM for non-cleared trades, or for portfolios consisting of many such trades, and is defined by the International Swaps and Derivatives Association (ISDA) [11]. It is a parametric model based on the VaR (Value at Risk) measure which estimates how much a portfolio might decrease in value during a set period of time within a given confidence interval - the larger the VaR, the larger the IM needs to be to cover the potential losses. Formally, SIMM has a margin requirement to meet a 99% confidence level over a 10-day *standard margin period of risk* [12]. This means that, in the event of a defaulting counterparty, the IM should have a probability of at least 99% to cover incurred losses during the first 10 days until the affected party is able to close out its financial instruments against the defaulting counterparty and re-hedge its resulting market risk [16].

To use SIMM, a market participant (such as a bank) needs to calculate and provide various input sensitivities of their portfolio for predetermined risk factors and product categories into a mathematical model provided by ISDA. The SIMM model is calibrated using market data in historical stress periods and its output is therefore reliant on ISDA centrally providing correlations and risk weights for the aforementioned risk factors and product categories [5, 17].

SIMM methodology

In the SIMM methodology, there are four so-called product classes:

- Interest Rates and Foreign Exchange (RatesFX)
- Credit
- Equity
- Commodity

and six risk classes:

- Interest Rate
- Credit (Qualifying)

- Credit (Non-qualifying)
- Equity
- Commodity
- FX

The final SIMM value is calculated as a sum over the four product classes p

$$\text{SIMM} = \sum_p \text{SIMM}_p. \tag{2.1}$$

The SIMM value for a product class p is given by

$$\text{SIMM}_p = \sqrt{\sum_i \text{IM}_i^2 + \sum_i \sum_{i \neq j} \psi_{ij} \text{IM}_i \text{IM}_j}, \tag{2.2}$$

where the summation of i and j is done over the six risk classes and ψ_{ij} is a constant correlation coefficient between risk classes. Finally, the margin IM_r for a risk class r is defined to be the sum of margins:

$$\text{IM}_r = \text{DeltaMargin}_r + \text{VegaMargin}_r + \text{CurvatureMargin}_r + \text{BaseCorrMargin}_r. \tag{2.3}$$

Because the portfolios in this thesis consist solely of FX instruments, only the RatesFX product class and the FX risk class need to be considered. This eliminates the summation over product classes (in 2.1) as well as the risk classes and the correlation terms (in 2.2), which leaves the much simplified SIMM calculation

$$\begin{aligned} \text{SIMM} = \text{SIMM}_{RatesFX} &= \sqrt{\sum_i \text{IM}_i^2 + \sum_i \sum_{i \neq j} \psi_{ij} \text{IM}_i \text{IM}_j} \\ &= \sqrt{\sum_i \text{IM}_i^2} \\ &= \text{IM}_{FX}. \end{aligned}$$

Finally, the IM_{FX} calculation further simplifies to the DeltaMargin_{FX} term in Equation 2.3 because the BaseCorrMargin term is only applicable to the credit risk class, and the VegaMargin and CurvatureMargin terms are only applicable to instruments that are subject to optionality, which the FX instruments in this

thesis are not. This leaves a portfolio $\mathbf{p}$ with sensitivities $[s_1, \ldots, s_n]$ with a final SIMM IM value of

$$\text{SIMM} = \text{IM}_{FX} = \text{DeltaMargin}_{FX} = \sqrt{\sum_k \sum_l \rho_{kl} f_{kl} w_k s_k w_l s_l} \ ,$$

where the summation is done over the unique currency pairs in the portfolio and uses their corresponding sensitivities s_k (DV01 values). The parameters ρ_{kl}, w_k and f_{kl} are constants and are left described in Section 8 of the official SIMM methodology [11]. The computational complexity of SIMM, with the limitations and simplifications made in this study, is $\mathcal{O}(Cn^2)$ where n is the number of sensitivities in the portfolio, and C is a constant associated with the cost of computing the parameters ρ_{kl}, w_k and f_{kl}.

2.1.4 CCP IM models

While SIMM has many advantages such as being based on a set of reliable and pre-defined assumptions and being publicly available, it also has disadvantages. It attempts to provide a general, 'one-size-fits-all' IM model for non-cleared trades which makes assumptions that might not always be sound or appropriate for larger market players such as CCPs. In such instances, CCPs can use internal scenario-based CCP IM models. Unlike SIMM, these are not publicly available and can be implemented to fit the specific needs of different CCPs. Even CCPs that use the same IM model framework type, such as VaR, can use drastically different parameters and thus get significant output differences [5]. However, what can be stated about the CCP IM model that is used in this thesis is that it follows the structure of scenario-based VaR methodology, as described in general terms below.

Scenario-based VaR models

Given a portfolio $\mathbf{p} \in \mathbb{R}^n$ consisting of n sensitivities and a set of m scenarios $\mathbf{q} \in \mathbb{R}^n$ consisting of observed market movements for the risk factors in $\mathbf{p}$, the PnL (Profit and Loss) of $\mathbf{p}$ is defined as the dot product

$$\text{PnL}(\mathbf{p}) = \mathbf{p} \cdot \mathbf{q}.$$

The PnL is thus simply the sum of multiplying the observed changes in a scenario with their corresponding sensitivities. A positive PnL therefore implies a profitable scenario for the portfolio, while a negative PnL implies a non-profitable scenario. By calculating the PnL for all m scenarios and sorting

them, one can choose the PnL of e.g. the 5% least profitable scenario (i.e. the VaR) as the IM for the portfolio. Other methods, such as Expected Shortfall (ES), would instead take an average of the PnLs of the 5% worst scenarios as the IM for the portfolio.

The computational complexity of the dot product for calculating the PnL of one scenario is $\mathcal{O}(n)$, and obtaining m PnLs is therefore $\mathcal{O}(nm)$. Finally, sorting the m PnL values has the standard $\mathcal{O}(m \log m)$ time complexity of optimal sorting algorithms. The computational complexity of a simple scenario-based VaR model is thus $\mathcal{O}(nm + m \log m) = \mathcal{O}(nm)$, making it linear in the size of the portfolio and in the number of scenarios. However, note that this is the minimal amount of work required for a scenario-based VaR model, and that implementations may be more complex.

As a quantile-based method where only the worst performing scenarios are relevant for the final IM value, considering all m scenarios in the IM computation amounts to redundant computations - computations that an ML algorithm would not have to perform once trained. On the other hand, evaluating a trained neural network with a total of W trainable parameters on a portfolio of n sensitivities has a computational complexity of $\mathcal{O}(Wn)$.

2.2 Artifical Neural Networks

Artificial neural networks (ANNs), or just neural networks, are machine learning models inspired by the structure and function of the human brain [18]. This section aims to give the reader a fundamental understanding of a specific type of ANN, namely the feedforward neural network (FNN), by describing its origins, developments and mathematical properties.

2.2.1 Linear regression and neural networks

To understand neural networks, one needs only understand basic linear regression. Recall the problem of multivariate linear regression: given a labelled dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ of N m-dimensional inputs, where each input $\mathbf{x}_i \in \mathbb{R}^m$ has corresponding label $y_i \in \mathbb{R}$, find a linear function

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m = \sum_{i=1}^{m} \beta_i x_i + \beta_0$$

that minimizes a chosen *loss function* for all points $\mathbf{x}$ in the dataset. An example of common loss function (for regression problems) is the residual sum of

squares

$$L_{RSS} = \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2,$$

which measures the sum of squared differences between the function's output $f(\mathbf{x}_i)$ and the expected output y_i for all observed data. The problem of linear regression thus requires finding the coefficients $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_m]$ that minimize the chosen loss function.

The simplest type of neural network is only a linear regression model whose output is passed through some nonlinear function, called an *activation function*. Such a network can therefore be expressed as

$$n(\mathbf{x}) = a(f(\mathbf{x})) = a(\sum_{i=1}^{m} \beta_i x_i + \beta_0),$$

where a is an activation function. Training the network n is therefore a similar problem to linear regression - both problems require finding the coefficients $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_m]$ that minimize the loss function. Analytical methods for finding the coefficients in a linear model, like the method of least squares, have been used since the 1800s by mathematicians like Legendre and Gauss [19], but no such analytical equivalent exists for finding the weights in a neural network [18]. Section 2.2.4 describes the most common way of training neural networks.

Geometrically, the linear regression model $f(\mathbf{x})$ represents a hyperplane in an m-dimensional space whose position relative to the origin is given by the constant coefficient β_0 and whose orientation is determined by the m remaining β-coefficients. In the setting of neural networks, the β_0-coefficient is commonly referred to as the *bias* term (or *threshold* term, as used in Section 2.2.2), while the other β-coefficients are typically called *weights*.

2.2.2 Single-layer perceptron

The concept of ANNs was first introduced by McCulloch and Pitts in 1943 where they modeled an artificial neuron to mimic the way neurons in the brain are activated to transmit electronic signals [20]. To this end, the McCulloch-Pitts artificial neuron was defined as a mathematical function which takes N binary input signals $x_1, \ldots, x_N$ and computes their weighted sum using the corresponding weights $w_1, \ldots, w_N$. The output of the neutron, known as its *activation*, is obtained by passing the weighted sum through an activation function; McCulloch and Pitts used the shifted Heaviside step function

$$g(x) = \begin{cases} 1 & x \geq t, \\ 0 & \text{otherwise.} \end{cases}$$

Simply put, if the weighted sum exceeds a threshold value t, the neuron is activated and outputs the value 1. Otherwise, it outputs 0. This computation is visualized in Figure 2.1.

The weight w_i represents the strength of the connection between input signal x_i and the neuron - the larger the weight, the more it contributes to the neuron's activation. Using the dot product notation $\mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{N} w_i x_i$, the McCulloch-Pitts neuron can succinctly be defined as

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} - t \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

In essence, the McCulloch-Pitts neuron is a binary classifier because it separates its input into one of two distinct classes, 0 and 1. It later came to be known as a single-layer perceptron (SLP), and was first implemented as a dedicated machine built by Frank Rosenblatt in 1958 [21]. In 1969, Minsky and Papert showed that the SLP was only able to solve linearly separable problems [22], which diminishes its usefulness as a binary classifier.

Geometrically, the goal of the SLP is to find a hyperplane (given by the weights $w_1, \ldots, w_N$ and the threshold value t) which separates inputs belonging to two different classes. However, many functions (such as the logical exclusive-or function), are not linearly separable and can therefore not be learned by an SLP - see Figure 2.2.

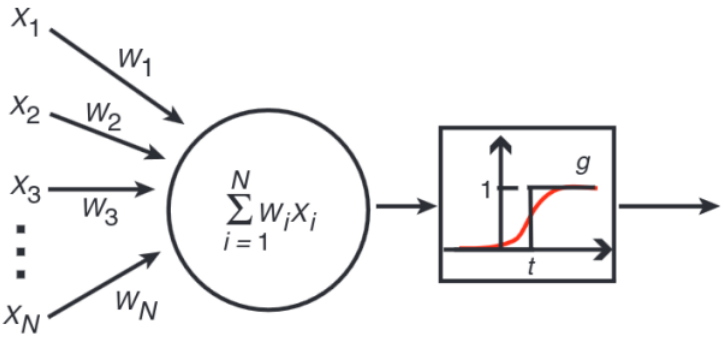


Figure 2.1: A visualization of the McCulloch-Pitts artificial neuron with two different example activation functions g - the Heaviside step function (shown in black) and a sigmoid function (shown in red) [18].

2.2.3 Multilayer perceptron

The multilayer perceptron (MLP) overcomes the SLP's limitation of only being able to learn linearly separable functions by extending the structure of the SLP with additional neurons (more commonly referred to as *nodes*). Instead of feeding the inputs directly into the output node (as in Figure 2.1), the MLP has an additional 'hidden' layer of nodes between the inputs and the output node, as shown in Figure 2.3a.

Figure 2.3 shows an MLP and an SLP, both of which have five inputs and one node in the output layer. However, the inputs of the MLP feed into the three nodes in the hidden layer instead of into the output node directly. Because the information in an MLP only flows in one direction - from the input layer, through one or many hidden layers, and into the output layer - they are also known as feedforward neural networks (FNNs).

Intuitively, the structure in Figure 2.3a relieves the output node of having to make the final classification (i.e. making the separating hyperplane) solely based on a linear transformation of the inputs. Instead, the nodes in the hidden layer create their own separating hyperplanes and pass these through a nonlinear activation function. These resulting nonlinear outputs serve as partial classifications and are sent on to the output node, which is now able to assemble

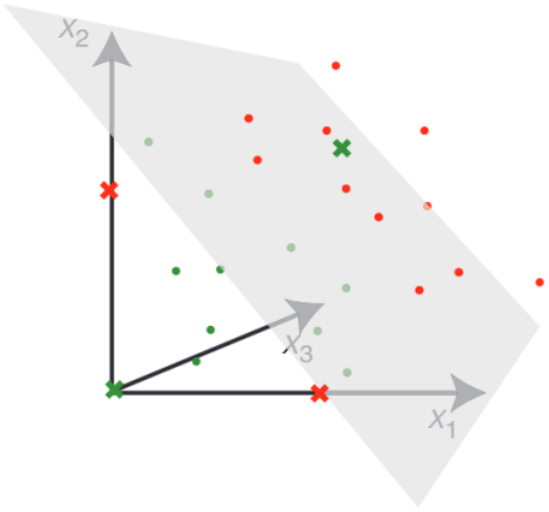


Figure 2.2: The red and green dots represent different inputs (x_1, x_2, x_3) to an SLP. The color of a point signifies its class (0 or 1), and the crosses define the exclusive-or function - these points cannot be separated by any plane in $\mathbb{R}^3$ [18].

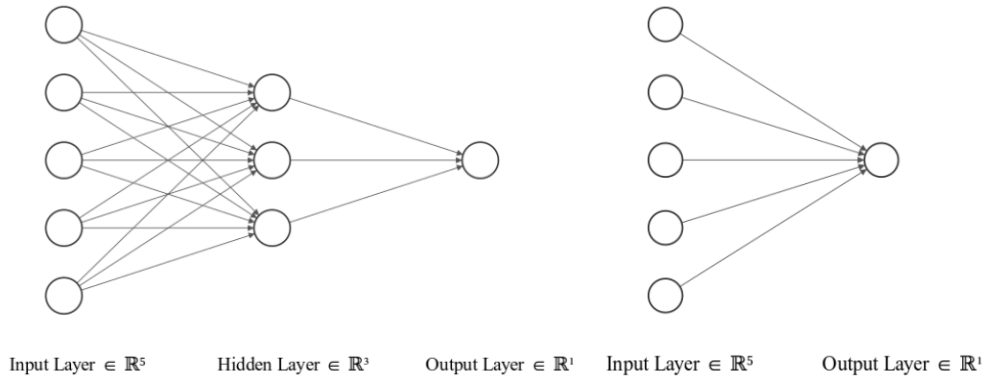
these partial classifications to a final nonlinear classification. By aggregating several linear functions with nonlinear activation functions this way, MLPs are able to learn complex and nonlinear functions [18]. Note that MLPs can also be used for regression instead of for classification if a continuous activation function is used instead of a discrete one.

Naturally, because the MLP has more nodes than the SLP, all of which require their own respective weights and threshold values, it is more difficult to train. Looking at Figure 2.3b, the SLP only has one node which requires learning one set of weights $w_1, \ldots, w_5$ and one threshold value t to create its separating hyperplane. However, the MLP in Figure 2.3a requires learning three distinct threshold values and three distinct sets of five weights $w_1, \ldots, w_5$ - one set for each node in the hidden layer. It also requires learning an additional three weights and a threshold value for the node in the output layer.

Indeed, how to efficiently train MLPs was a widely researched problem for two decades until the backpropagation algorithm (described in Section 2.2.4 below) was standardized by Werbos in 1982 and further popularized by Rumelhart et al. in 1986 [23, 24].

2.2.4 Training neural networks

The goal of training a neural network $f(\mathbf{x}; \mathbf{W})$ is to find its defining parameters $\mathbf{W}$ which allow it to learn some underlying function or pattern from the dataset



(a) (b)

Figure 2.3: (**a**) An MLP with a hidden layer consisting of three nodes between the inputs and the output layers. (**b**) An SLP where the inputs feeds directly into the output layer (equivalent to the McCulloch-Pitts neuron in Figure 2.1).

$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ it is trained on. To that end, an error measure for f is defined in the form of a loss function (example of a loss function is provided in Section 2.2.1). First, let $\lambda\big(y, f(\mathbf{x}; \mathbf{W})\big)$ denote the partial loss of f when applied to a single input $\mathbf{x}$ compared to its expected output y. The problem of training f for a dataset D can then be formalized as minimizing the total loss function

$$L(f(\mathbf{x}; \mathbf{W})) = \sum_{(\mathbf{x},y)\in D} \lambda\big(y, f(\mathbf{x}; \mathbf{W})\big), \tag{2.4}$$

which is simply the sum of the partial losses of f for all points in D.

Gradient descent

The most commonly used training algorithms are variations of the gradient descent (GD) algorithm such as stochastic gradient descent (SGD) or other further improvements of these algorithms [25]. GD is an iterative optimization algorithm for finding a local minimum of any differentiable function. The idea behind GD is that if a function $f(\mathbf{x})$ is differentiable around $\mathbf{x}$, then the direction in which $f(\mathbf{x})$ decreases fastest is given by the negative gradient of f at $\mathbf{x}$, denoted as $-\nabla f(\mathbf{x})$. At iteration i, GD finds the next point $\mathbf{x}_{i+1}$ to evaluate by updating the current point $\mathbf{x}_i$ with the following calculation

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \eta \nabla f(\mathbf{x}_i), \tag{2.5}$$

where $\eta \in \mathbb{R}_+$ is a small, positive value called the *learning rate*. With a sufficiently small learning rate, GD thus ensures $f(\mathbf{x}_{i+1}) \leq f(\mathbf{x}_i)$, i.e. the training converges towards a local (or global) minimum. This leaves one important question unanswered - how is the gradient $\nabla f(\mathbf{x})$ computed for complex functions like neural networks?

Gradient descent with backpropagation

Consider using GD to train a generic FNN f consisting of n layers $f^{(1)}, \ldots, f^{(n)}$, expressed as

$$f(\mathbf{x}; \mathbf{W}) = f^{(n)}(f^{(n-1)}(\ldots(f^{(2)}(f^{(1)}(\mathbf{x}))))),$$

where $\mathbf{x}$ is an input vector and $\mathbf{W}$ are the network's parameters, i.e. the weights and the biases of the nodes in each layer $f^{(i)}$ of the network.

To do so, one must minimize the loss function $L(f(\mathbf{x}; \mathbf{W}))$ shown in Equation 2.4. This requires finding the gradient $\nabla_{\mathbf{W}} f(\mathbf{x}; \mathbf{W})$ by differentiating

$f(\mathbf{x}; \mathbf{W})$ with respect to the parameters $\mathbf{W}$ of the network. The most common way of computing the gradient $\nabla_{\mathbf{W}} f(\mathbf{x}; \mathbf{W})$ is using the backpropagation algorithm [26]. It is an efficient dynamic programming algorithm which computes the partial derivatives of the loss function with respect to the weights in each layer - iterating backwards from the last layer $f^{(n)}$ to the first layer $f^{(1)}$ by applying the Leibniz chain rule. Once the gradient is calculated, the improved network weights can be calculated using Equation 2.5.

2.3 Overfitting

Section 2.2.4 described the training process which only considers how to train a model to fit a given dataset. However, it does not address a fundamental problem in machine learning; that of *overfitting*. The difficulty of machine learning lies not in training models that perfectly fit observed data, but in using the observed data to train models which generalize well and are useful when applied to new data.

Overfitting occurs when a model is overspecified, i.e. when it has too many free parameters for the size of the dataset it is trained on [18]. This results in the model fitting the training data too closely and learning potential noise in the data, which reduces the model's ability to generalize to new data. The opposite can also be true - if a model has fewer parameters than would be required to properly fit the data, it is said to be *underfitted*.

2.3.1 Cross-validation

Cross-validation is one of the most common validation methods for evaluating a machine learning model's generalization capabilities and for preventing overfitting [27]. The idea of cross-validation is to separate the dataset into a *training set* which is used for training the model, and a *test set* which is used to evaluate how the trained model performs on data it has not seen before. To reduce variance, this process is repeated multiple times so new models are trained and tested on different subsets of the dataset. The average of these models' performances on their respective test sets provide an estimate of the final model's performance.

There are different resampling strategies for splitting the data into training and test sets. One such strategy is *k-fold random subsampling*, where the training set is chosen by randomly sampling typically around 70%-90% of the dataset, and the remaining data becomes the test set. This is done a total of k times.

Another strategy is *k-fold cross-validation* where, unlike the random subsampling, the test set is guaranteed to be unique in each of the k iterations. This is done by partitioning the dataset into k disjoint subsets and training a model using each of the k subsets as a test set while the remaining $k-1$ subsets are used for training, as shown in Figure 2.4.

2.3.2 Cross-validation with a validation set

In the context of training ML models, one seldom only trains a single model and evaluates it using cross-validation because the model performance can be improved significantly if its hyperparameters are optimized (more on this in Section 2.6). To be able to train multiple models with different hyperparameters and compare their performances, these must be evaluated on data which is neither training nor test data. For this purpose, a part of the training set, called the *validation set*, is set aside. This allows multiple different models to be trained on the training set and evaluated on the validation set, and once a final best-performing model is found, it can be evaluated on the unseen test set.

2.4 Decision Trees

Decision trees are tree-structured ML models that can be used for both regression and classification tasks [29]. A decision tree is an intuitive model for representing the decision-making process of evaluating multiple conditions to

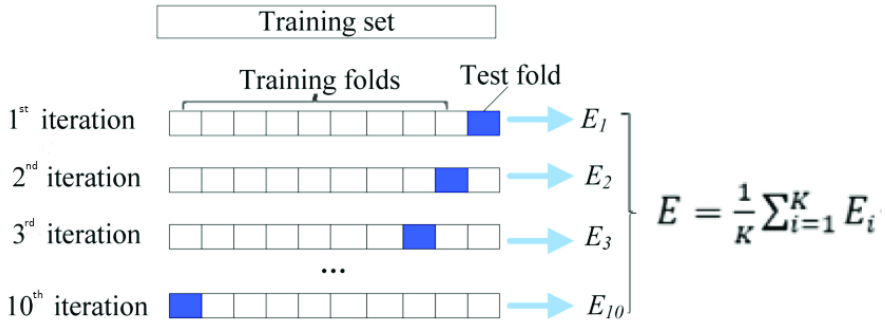


Figure 2.4: 10-fold cross-validation. Ten models are trained on 90% of the dataset and tested on the remaining 10%. Each model's error on the test set is denoted E_i, and the final model's error E is estimated as an average over E_i [28].

make a specific decision. Figure 2.5 illustrates a decision tree for a classification task (a *classification tree*), where each observed data point (x_1, x_2) belongs to one of two classes.

The topmost node in the tree is called the *root node*. The nodes beneath the root that branch out into smaller subtrees are called *branching nodes* or *decision nodes*, and the bottom nodes are called *leaf nodes* or simply *leaves*. The leaves of a decision tree represent its final predicted class or label for an input point, and the path from the root node to a given leaf thus signifies the conditions that the input point satisfies in order to reach the leaf.

2.4.1 Constructing decision trees

Training (or constructing) a decision tree requires determining the conditions in the branching nodes and the labels in the leaves so that the final tree minimizes some error metric on the dataset. This is done by recursively partitioning the original dataset into smaller subsets by separating (*splitting*) the inputs in the dataset based on their values for a specific input feature [30].

For example, the root node in Figure 2.5 splits the dataset on the feature x_1 so that inputs satisfying $x_1 < 5$ are assigned to the left subtree while the rest are assigned to the right. The inputs assigned to the left have reached a leaf and are therefore classified as one class, while the inputs assigned to the right subtree are split an additional time before reaching a leaf node and receiving a classification. The specific split $x_1 < 5$ is determined by iteratively comparing splits on different features and different values of those features, and choosing

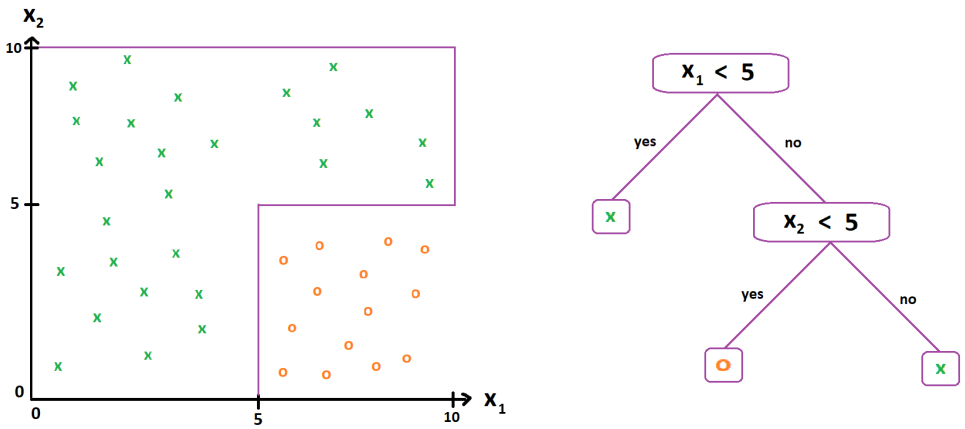


Figure 2.5: Example of a classification tree for classifying two-dimensional points (x_1, x_2) as one of two classes.

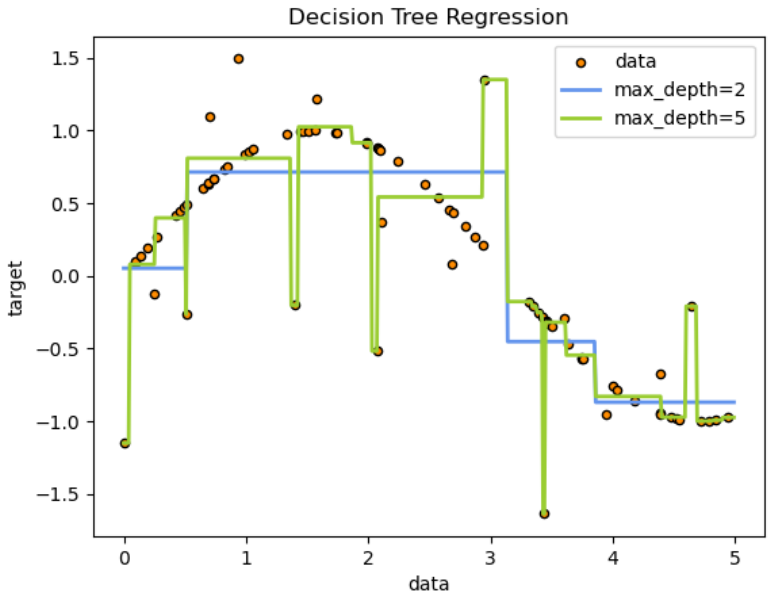


Figure 2.6: One-dimensional regression problem where the 'data' feature is used to predict the scalar 'target' label. The blue and green graphs show the predictions for regression trees with maximum depths set to 2 and 5, respectively [31].

the split which minimizes some error metric on the resulting tree. Informally, this process finds the most significant or relevant feature to split the dataset on so the resulting tree most accurately distinguishes its predicting labels.

2.4.2 Preventing overfitting

In theory, by splitting the data into smaller and more specific subsets, a decision tree can be grown to its maximum capacity where each unique input point is assigned its own leaf. However, this likely results in an overly complex and overfitted tree which is able to perfectly predict, or rather *memorize*, all labels for the training data but which generalizes poorly to new data. The construction of decision trees is therefore terminated at various stopping criteria such as when a maximum tree depth is reached or when a branching node has few enough remaining input points [30].

Figure 2.6 visualizes the problem of overfitting in regression trees for a one-dimensional regression problem. The green graph shows the predictions for an overfitted regression tree where the higher maximum depth causes the

tree to learn from noise in the data. The blue graph, on the other hand, is significantly simpler and only spans four different intervals for the prediction variable, and predicts the average of the target values for all data points in each interval.

Despite the ability to control the complexity of decision trees by using various hyperparameters, they can still be overly sensitive to training data resulting in trees with high variance [29]. To avoid overfitting, methods such as *tree pruning* are often used. Tree pruning is the process of growing a large decision tree before incrementally removing non-critical parts of the tree as identified by cross-validation [30].

2.5 Bootstrap aggregating

Effective alternatives to decrease the variance and improve the performance of decision trees are ensemble learning methods such as *bootstrap aggregating*, or simply *bagging* [32]. Bagging combines the outputs of multiple ML models by either averaging their outputs (in regression problems) or choosing the majority class vote from the predictions of each model (for classification problems). Each model is trained on a distinct, *bootstrapped* dataset generated by uniformly sampling from the original dataset with replacement. This means that bootstrapped datasets can contain duplicate data points, which allows for training a large number of models on diverse datasets, regardless of the original dataset's size.

2.5.1 Random Forests

A random forest (RF) is a modification of bagged decision trees which further decreases variance by introducing additional randomness when constructing each tree. When constructing the individual trees in an RF on data with m features, the trees only consider $k \leq m$ randomly chosen features as candidates for splitting at each node, instead of all m. This reduces the correlation between the trees in the RF and thereby reduces the variance of the model as a whole [32].

2.6 Hyperparameter optimization

Hyperparameters are parameters used to configure the structure of an ML model and must be chosen prior to training. The choice of hyperparameters is

often critical and they must therefore be tuned to achieve optimal performance of an ML model [33, 34]. This section describes two algorithms for hyperparameter optimization (HPO) as well as relevant hyperparameters for FNNs and RFs.

2.6.1 Grid Search and Random Search

Grid search (GS) and random search (RS) are two of the most common HPO algorithms. Both algorithms train and evaluate multiple models using different hyperparameters to find the hyperparameters which result in the best-performing model. What differentiates them is that GS performs an exhaustive search of all hyperparameter values in a specified range of values for each hyperparameter, whereas RS chooses hyperparameter values randomly from these ranges [35].

2.6.2 Hyperparameters of FNNs

Depth and width

The *depth* of a neural network refers to how many layers it has (not counting the input layer). The term *deep learning* originates from this; the training of neural networks with many layers [25].

The number of nodes in each layer is called the layer's *width*. The depth and width of the network are commonly called *structural hyperparameters* because they determine the network's structure, and they are among the most important hyperparameters to optimize to ensure the network is sufficiently complex to avoid both overfitting and underfitting.

Optimizers

The optimization algorithm used during training is called the *optimizer*. Most optimizers are variations and improvements of the gradient descent algorithm described in Section 2.2.4. The main problem of the basic GD optimizer is that calculating the gradient requires processing the entire dataset in each iteration, which is computationally expensive for large datasets. In these contexts, the term *epoch* refers to how many iterations the optimizer requires before every training sample is used in updating the model parameters. For GD, an epoch is thus only a single iteration because GD considers all training samples to calculate the gradient.

However, instead of calculating the exact gradient in each iteration, one can instead choose to approximate it by using fewer samples. An epoch thus requires more iterations, depending on how many samples are considered in each iteration. Stochastic gradient descent (SGD) approximates the gradient using a single randomly chosen data point, making each iteration significantly less computationally expensive at the cost of rough gradient approximations.

Mini-batch stochastic gradient descent uses a fixed number of data points (called the *batch size)* to calculate the gradients, thus resulting in slightly more expensive and accurate gradient calculations than SGD, but not as accurate or expensive as GD.

Additionally, these variants of GD all suffer from the problem of finding a suitable learning rate. As described in Section 2.2.4, GD only guarantees convergence for sufficiently small learning rates, but too small learning rates result in the algorithm requiring many more iterations, and consequently, much more time to converge [25].

SGD was improved by the introduction of the Adaptive Gradient Algorithm (AdaGrad) which eliminates the need to manually tune the learning rate [36]. Instead of using a fixed learning rate for the entire training duration, AdaGrad dynamically assigns each parameter θ its own learning rate η_θ based on historical gradients from previous iterations and a global learning rate η (typically set to 0.01). However, AdaGrad still has the disadvantages of having to choose η, and it also suffers from inefficient parameter updates as training time increases [37].

The Adam optimizer is a further improvement of AdaGrad based on adaptive estimation of the first and second order moments of the gradients [38]. It also uses parameter-specific learning rates which are adjusted using stored values of exponentially decaying averages of previous gradients and their squares [37].

Activation functions

As explained in Section 2.2.3, MLPs are only able to learn nonlinear functions if a nonlinear activation function is used. To successfully train an MLP with gradient descent however, the activation functions should also have other desirable properties. For example, the binary Heaviside function

$$f(x) = \begin{cases} 1 & x \geq 0, \\ 0 & \text{otherwise}, \end{cases}$$

always differentiates to 0 except at $x = 0$ where it is non-differentiable. Because it never differentiates to any non-zero value, GD is unable to update the

model weights, making training impossible.

To avoid this, one can use continuously differentiable functions with non-zero derivatives such as the logistic function

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

or the hyperbolic tangent function

$$f(x) = \tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$

For deeper MLPs however, these activation functions cause a similar problem to the Heaviside function, called *the vanishing gradient problem* [39]. While they do not differentiate to 0, their derivatives converge to 0 as x tends to ∞ and $-\infty$. Such activation functions are said to *saturate* in both directions. This can result in vanishingly small gradients which either significantly slows down or completely halts the training process [40].

The current most popular activation function in deep neural networks is the Rectified Linear Unit (ReLU) [41]

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

which simply returns the input value itself for non-negative inputs and 0 for negative inputs. It is computationally more efficient than the previously mentioned activation functions, and it avoids the vanishing gradient problem because it only saturates in one direction, i.e. only as x tends to $-\infty$ [39].

Initialization strategy

So far, this chapter has mentioned how the weights and biases are updated to train an MLP, but it has not addressed how these are initialized. One such strategy is the *He initialization strategy* - it initializes all biases to zero and initializes each weight w at layer l by sampling from a zero-mean Gaussian distribution with a standard deviation $\sqrt{2/n_l}$, where n_l is the number of nodes in layer l [42]. It was the first initialization strategy to surpass human-level image classification on the ImageNet dataset, and it is therefore considered state-of-the-art, especially when used with the ReLU activation function for which it was designed.

2.6.3 Hyperparameters of RFs

Due to the simple and visually intuitive nature of decision trees, the hyperparameters of RFs are significantly easier to describe and understand than the

hyperparameters of FNNs. Below are short descriptions of some relevant hyperparameters for RFs and decision trees (using parameter names from Scikit-learn documentation [43]):

- **Random forest-specific hyperparameters**:
  - 'n_estimators': The number of decision trees in the forest
  - 'max_features': The number of randomly chosen features each tree considers when splitting a node
  - 'bootstrap': A boolean which determines if the dataset is bootstrapped or if the original dataset is used when building each tree
- **Decision tree-specific hyperparameters:**
  - 'max_depth': The maximum depth of the tree
  - 'min_samples_split': The minimum number of input points required to split a node
  - 'min_samples_leaf': The minimum number of input points required at a leaf node. If splitting a node would create a leaf with fewer input points, the split is not considered.

2.7 Related work

The most closely related research to this project is the study on initial margin simulations with deep learning by Ma et al. [14]. The purpose of the study is to use deep neural networks, specifically FNNs, for portfolio IM optimization.

They show that FNNs are practical and effective at approximating the SIMM model, which was used to generate the IM training labels. Additionally, because SIMM is deterministic, the training data has no noise which minimizes concerns of overfitting. Even with a relatively small FNN consisting of three hidden layers with eight nodes per layer, the authors are able to obtain IM estimations with a mean absolute percentage error of 3% on test portfolios.

This project extends the work of Ma et al. by also training a neural network and a random forest model on an additional dataset where the IM labels are generated using a CCP IM model instead of using SIMM, effectively creating a separate learning problem. This enables a comparison of the application of FNNs on different IM models and sheds light on the differences in data and complexity between SIMM and the CCP IM model.

A notable difference in the study by Ma et al. is that their network is trained on portfolios consisting of vanilla interest rate swaps and cross-currency swaps instead of the FX instruments that comprise the portfolios in this study.

Another similar study is that of Villarino and Rodríguez [44]. In it, the authors optimize the structural hyperparameters of a self-normalizing neural network to estimate IM for simple portfolios consisting solely of interest rate swaps, with training labels obtained from the SIMM model [44]. They find that their chosen deep learning model provides good approximations of the SIMM model. The results showed that a network with a depth of four and with the largest considered width of 128 nodes per layer gave the best performance.

This project follows the authors' suggestions for future work and builds a neural network to compute IM for real portfolios using SIMM as well as a CCP IM model using a scenario-based VaR methodology.

This study considers the portfolio sensitivities as given inputs and aims to approximate initial margin models such as SIMM and scenario-based VaR models using neural networks. However, in the context of IM optimization where dynamic IM is calculated in Monte Carlo simulations (as described in 2.1.2), the sensitivities also need to be recomputed in each time step of the simulation which is typically more expensive than evaluating SIMM.

Thus, the calculation of sensitivities also becomes subject to optimization in such simulations. In a study by Zeron and Ruiz, Chebyshev tensors are used to approximate the portfolio sensitivities which are then used as inputs to calculate IM with SIMM in each time step. A tensor in this context simply means a set of points $x_1, \ldots, x_n$ with a corresponding set of associated real values $v_1, \ldots, v_n$. When these points are chosen as the projection of equidistant points on the upper half of the unitary circle onto the real line, they are knows as Chebyshev points, and the resulting tensor would be called a Chebyshev tensor. A Chebyshev tensor defines a unique interpolating polynomial, aptly named a Chebyshev interpolant, which interpolates each Chebyshev point x_i at each value v_i. Chebyshev tensors, or rather their interpolants, are appoximators for analytics functions and have strong convergence properties and allow for efficient evaluation with high accuracy once built. [13]

While the problem of calculating sensitivities in dynamic IM optimizations was not included in the scope of this thesis, the results obtained by Zeron and Ruiz are nonetheless relevant because they show Chebyshev tensors outperform both regression methods such as the ones proposed by Chan et al [45], as well as deep learning models for calculating dynamic IM in terms of accuracy,

speed and ease of implementation.

Theoretically, one could combine the computation of dynamic sensitivities with Chebyshev tensors and feed them into an FNN trained on pre-computed sensitivities, such as the FNN in this thesis is, to further improve the efficiency of computing dynamic IM in the context of IM optimization. One potentially important difference, however, is that the financial instruments used in the study by Zeron and Ruiz are interest rate swaps and swaptions, which are inherently different from the FX instruments used to train the models in this study. To combine the methods in the way suggested above, the FNN would have to be trained on the sensitivities obtained from the Chebyshev tensors and not on any arbitrary sensitivities for different financial instruments.

On the use of neural networks for computing IM, the authors point out one of the main challenge when using machine learning methods - that of hyperparameter optimization. They mention that, in some cases, simple networks can provide sufficiently accurate results (as in [14] and [44]) while other times, the process can be more cumbersome and might not yield satisfactory models. With this in mind, the hyperparamter optimization in this study was conducted algorithmically and rigorously so as to provide a measure of the importance and effect of hyperparameter optimization for neural networks in this learning problem.

Chapter 3

Method

This study is a quantitative comparison of two different supervised ML models - feedforward neural networks (FNNs) and random forests (RFs) - applied to the estimation of two different models for calculating IM, SIMM and a proprietary CCP IM model. The goal is to find the best-performing ML model of the two, when applied to each of the models for IM calculation. To this end, the project can be divided into three separate tasks that are executed, and described below, in order.

3.1 Data

To solve the supervised machine learning problem of estimating portfolio IM with different IM models, the first step is to obtain a sufficiently large labeled dataset for each IM model in question. Depending on the input dimensionality (the number of features) for a given problem, neural networks can require thousands or tens of thousands data points to converge. However, since this project uses real portfolio data which typically is not public information, it is difficult to obtain a sufficient number of real portfolios.

To this end, OSTTRA have provided 40 portfolios of large, anonymous banks from which a desired number of portfolios can be generated - this is described in detail in Section 3.1.2. OSTTRA have also provided 2500 unique scenarios and access to a proprietary CCP IM model of an anonymous commercial CCP in the form of an executable binary file, as well as access to their internal SIMM implementation in C++.

3.1.1 Understanding the data

A dataset input $\mathbf{x} \in \mathbb{R}^d$ is a portfolio comprised of various FX instruments, represented as a d-dimensional vector of sensitivities (DV01 values). The dataset IM labels $y \in \mathbb{R}$ are obtained by evaluating a portfolio $\mathbf{x}$ on the 2500 provided scenarios, using an IM model - which, in this case, is either SIMM or the proprietary CCP IM model. However, the sensitivities that comprise the portfolios $\mathbf{x}$ are expressed differently for the two IM models, which requires generating separate portfolios, and consequently separate datasets, for each model.

CCP IM portfolios

For the CCP IM model, a portfolio $\mathbf{x}$ is represented by its sensitivities to the exchange rates of 26 unique currency pairs that its constituent FX instruments operate on. Each currency pair, in turn, consists of a corresponding spot rate and 10 forward rates covering a time span of between one week and five years in the future - see Table 3.1.

Exchange rate \ Currency pair		EUR/USD	SEK/USD	...	GBP/EUR
1 day	(spot rate)	118 000	-86 000	...	3 200 000
1 week	(forward rate)	620 000	-224 000	...	0
1 month	(forward rate)	161 000	14 000	...	-120 000
⋮		⋮	⋮	...	⋮
5 years	(forward rate)	0	0	...	0

Table 3.1: Example of an anonymous bank's portfolio containing 286 (rounded) DV01 values for 11 exchange rates of 26 currency pairs.

A portfolio for the CCP IM model is thus a 286-dimensional vector of sensitivities to the spot and forward rates for each of the 26 currency pairs (obtained by flattening the 11×26-dimensional matrix in Table 3.1).

SIMM portfolios

For SIMM, a portfolio $\mathbf{x}$ is represented by the sensitivities to the exchange rates of 50 unique currencies against the dollar. As such, they can be expressed as

50-dimensional vectors of sensitivities - as shown in Figure 3.2 - and are thus comparatively simpler than the CCP IM portfolios.

Currency pair	EUR/USD	USD/USD	SEK/USD	...	GBP/USD
DV01	2 000 000	0	-500 000	...	64 000

Table 3.2: Example of an anonymous bank's portfolio containing 50 (rounded) DV01 values for the exchange rates between 50 different currencies and USD.

3.1.2 Data generation

The 40 real bank portfolios used in this project consist of 24 SIMM portfolios and 16 CCP IM portfolios. This section describes, in three steps, how these bank portfolios are used to generate separate datasets for their respective portfolio type.

The first two steps are performed separately on the SIMM *and* CCP IM portfolios, while the last step is *only* performed on the CCP IM portfolios to generate additional portfolios of that type. This generation results in a SIMM dataset of size 57500 and a slightly larger CCP IM dataset of size 59100. The real portfolios are excluded from their respective datasets, and the generated portfolios are split into separate sets for training, validation and evaluation, as described in Section 3.3.

Step 1 - sampling from a joint portfolio distribution

The provided bank portfolios are used to calculate the means $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_m]$ and standard deviations $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_m]$, where μ_i and σ_i are the mean and standard deviation of feature i. These means and standard deviations are then used to sample $20\ 000$ normally distributed portfolios $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ to incorporate feature information from all of the provided bank portfolios.

Step 2 - sampling from uniform portfolio distributions

To avoid the risk of overfitting and training on a dataset generated exclusively based on a limited number of real portfolios, the dataset is complemented by a set of $37\ 500$ uniformly distributed portfolios. The features in these portfolios are sampled in equal parts from the three uniform distributions $U(-10^5, 10^5)$, $U(-10^6, 10^6)$ and $U(-10^7, 10^7)$. The motivation for using these distributions is to simulate the portfolios of small and large market players, and thus provide learning examples for estimating both small and large IM values. Finally, many

of the DV01 values in a portfolio were randomly set to zero to mimic most real portfolios that are not sensitive to all exchange rates, but rather only to a handful.

Step 3 (only for CCP IM) - sampling from individual portfolios

To compensate for the fact that the SIMM portfolio generation is done using 50% more real portfolios than the CCP IM portfolio generation, this step aims to incorporate more of each individual CCP IM portfolio into that dataset. To this end, an additional 100 portfolios are generated for each of the 16 CCP IM portfolios by making large uniform perturbations to each sensitivity in the portfolio. Specifically, for each bank $B \in [1, 16]$ with the portfolio $\mathbf{x}^B = [x_1^B, \ldots, x_m^B]$, 100 points $\mathbf{x}'$ are generated as

$$\begin{aligned} &\mathbf{x}' = [x_1, \ldots, x_m], \\ &x_i = x_i^B + U(-Cx_i^B \ , \ Cx_i^B) \quad \forall i \in [1, m], \end{aligned}$$

where $U(a, b)$ denotes a uniformly distributed perturbation between a and b, and the constant C determines the relative size of the perturbation to the original feature value x_i^B. The uniform distribution and the chosen constant $C = 0.7$ guarantee that 50% of the generated features x_i differ from the original features x_i^B by at least $\frac{C}{2} \cdot 100\% = 35\%$, which was deemed to sufficiently diversify the original portfolio.

3.2 Implementation

With labeled datasets ready for both IM models, the second step of the project is to implement the FNN and RF models. This was done using Python (version 3.9) for its extensive and well-documented ML libraries. The FNN was implemented using the Keras library [46] and optimized using the KerasTuner framework [47], while the RF model was implemented and optimized using the Scikit-learn library [48].

With regards to hardware, Google Colab was used throughout the project to utilize its effective GPU/TPU resources [49] for model training, while data generation and model evaluations was done locally on a laptop (Dell XPS 13 9380, Intel Core i7 CPU, integrated GPU) using the PyCharm IDE for its convenient data visualization functionality [50]. The CCP IM and SIMM models were evaluated on a remote machine using an NVIDIA Tesla V100 Tensor Core GPU.

3.3 Model optimization and evaluation

The third and final step of the project is optimizing and evaluating the FNN and RF models on both datasets. This was done using 3-fold cross-validation with validation sets, as described in Section 2.3.2. The datasets were thus split into training, validation and test sets that contain 81%, 9% and 10% of the total data points.

The hyperparameter optimization was mostly done using automated algorithms, but some hyperparameters were simply optimized manually or not at all. Both models are evaluated using the same loss function - the Mean Absolute Percentage Error (MAPE):

$$\text{MAPE} = \frac{100\%}{N} \sum_{i=1}^{N} \left| \frac{y_i - \hat{y}_i}{y_i} \right|,$$

where N is the number of data points, y_i is the IM label of the i'th point and $\hat{y}_i$ is the model's IM estimation for the i'th point.

The FNN in this project uses the *He* initialization strategy, the Adam optimizer and the ReLU activation function. These hyperparameters are chosen because of their respective strengths (as described in Section 2.6.2), and due to limitations in training times, alternatives were *not* considered during optimization. Instead, the hyperparameter optimization focused on the structural hyperparameters (the depth and width) of the FNN. These were optimized using a combination of the HPO algorithms grid search (GS) and random search (RS). RS was used to perform a coarse-grained search through a large range of widths and depths and find promising intervals where the model performs well. The hyperparameter search was then complemented using GS to perform a more fine-grained search through the smaller, more promising intervals of hyperparameter values identified by RS.

The RF hyperparameters described in Section 2.6.3 were all optimized using the same methodology of using RS to perform a coarse-grained search followed by one or more fine-grained searches using GS.

Chapter 4

Results

This chapter presents the best-performing feed-forward neural network and random forest models on both the SIMM and CCP IM datasets, evaluates their performance on unseen portfolios, compares their execution time to their respective IM models and describes the hyperparameter optimization process.

4.1 Best-performing models

4.1.1 FNN results

The three best-performing FNNs found in this study and their corresponding test errors on the CCP IM and SIMM datasets are shown below in Tables 4.1 and 4.2.

Depth	Width (nodes per layer)	Test error
3	600, 1000, 200	1.50%
2	600, 600	1.52%
2	600, 800	1.54%

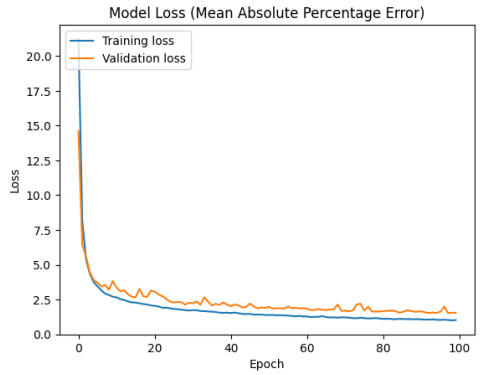
Table 4.1: Test error for the three best-performing models on the CCP IM dataset

Depth	Width (nodes per layer)	Test error
3	3000, 1000, 500	0.490%
3	3000, 2000, 500	0.492%
3	3000, 500, 100	0.503%

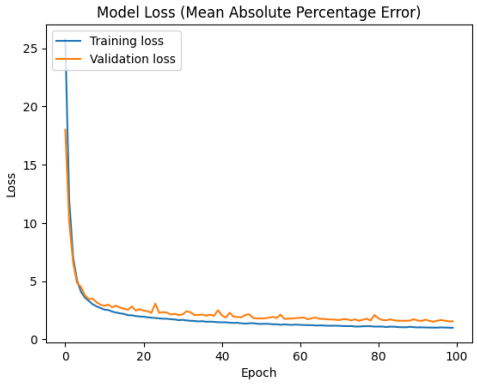
Table 4.2: Test error for the three best-performing models on the SIMM dataset.

The best-performing FNNs for both datasets thus have three hidden layers, with the best FNN for the SIMM dataset being wider with 3000, 1000 and 500 nodes per layer and a total of $W = 3\,655\,001$ trainable parameters while the FNN for the CCP IM dataset has 600, 1000 and 200 nodes per layer with a total of $W = 973\,601$ trainable parameters.

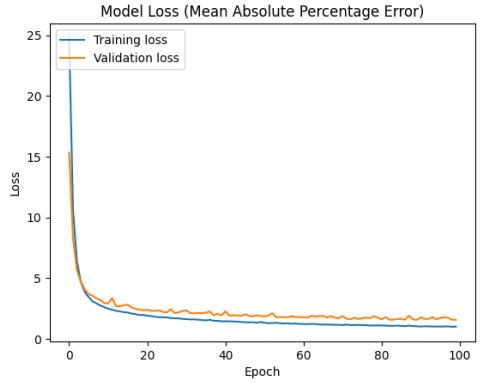
These models were obtained using hyperparameter optimization methods (described in detail in Section 4.4), and then re-trained until convergence for 100 epochs. The training and validation losses of the models are plotted during each epoch in Figures 4.1 and 4.2.



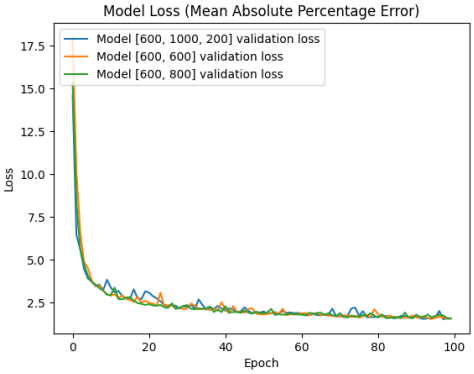
(a) Training and validation loss for FNN with widths $[600, 1000, 200]$.



(b) Training and validation loss for FNN with widths $[600, 600]$.

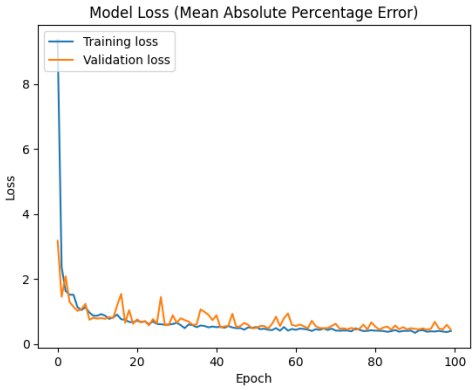


(c) Training and validation loss for FNN with widths $[600, 800]$.

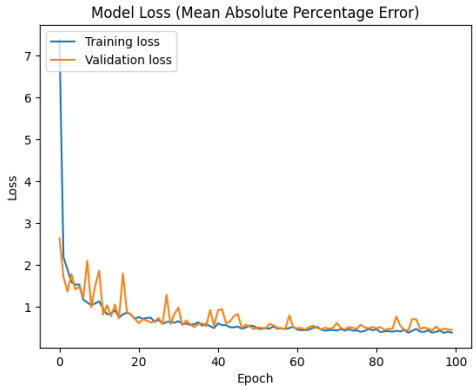


(d) Validation losses of the three models.

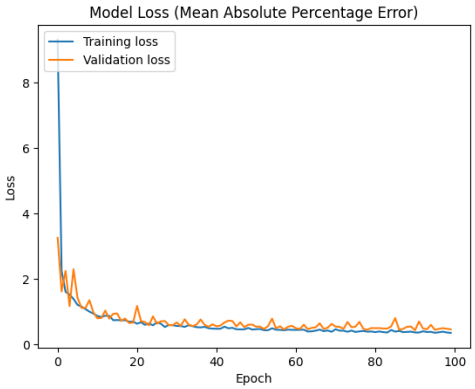
Figure 4.1: Training and validation loss plots for 100 epochs for three FNNs on the CCP SIMM dataset, with a side-by-side validation loss comparison.



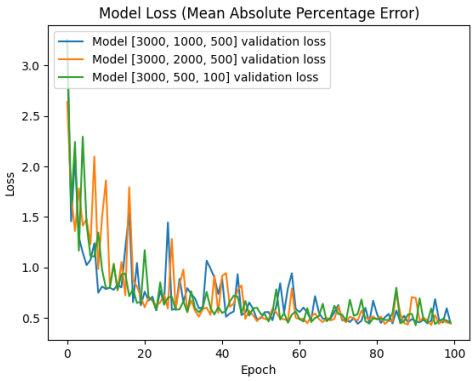
(a) Training and validation loss for FNN with widths $[3000, 1000, 500]$.



(b) Training and validation loss for FNN with widths $[3000, 2000, 600]$.



(c) Training and validation loss for FNN with widths $[3000, 500, 800]$.



(d) Validation losses of the three models.

Figure 4.2: Training and validation loss plots for 100 epochs for three FNNs on the SIMM dataset, with a side-by-side validation loss comparison.

4.1.2 Random forest results

The best-performing random forest model for the CCP IM and SIMM datasets (obtained using the hyperparameter optimization described in Section 4.4) are summarized in Table 4.3 below.

Dataset / HP name	CCP IM	SIMM
n_estimators	50	100
max_features	32	50
max_depth	100	∞ (no max depth)
min_samples_split	10	2
min_samples_leaf	1	1
bootstrap	True	True
Test error:	12.3%	17.3%

Table 4.3: The best-performing random forest model hyperparameters on the CCP IM and SIMM datasets, and their corresponding test errors.

4.2 Evaluating the data generation

To evaluate the chosen data generation methods (described in Section 3.1.2), the 16 real CCP IM portfolios and the 24 SIMM portfolios were excluded from the training set and used to test the best-performing FNN models separately. The results were consistent with the models' test errors of 1.50% and 0.490% as they achieved average errors of 1.53% and 0.502% on the real bank portfolios.

4.3 Efficiency

This project could not conduct a rigorous efficiency comparison between the FNNs and the two IM models on the same hardware because the CCP IM model is proprietary software only available through a remote host. While the SIMM model is public and possible to re-implement, doing so would fall beyond the scope of this project. What can be asserted given these restrictions, however, is that both IM models were run on significantly more powerful hardware which took between 30 and 60 seconds to output IMs for 2500 portfolios. Contrarily, obtaining the corresponding IM values by evaluating the trained FNNs in this project was almost six times faster using a laptop, as shown in Table 4.4.

Model \ Dataset	CCP IM	SIMM
FNN model	9.40 s	6.25 s
IM model	54.1 s	35.6 s

Table 4.4: The execution times of calculating IM for 2500 portfolios for each dataset (SIMM and CCP IM) using its corresponding IM and FNN model.

4.4 Hyperparameter optimization

4.4.1 FNN optimization

The first rounds of optimizations on the depth and width of the FNN were done using random search (RS) and 3-fold cross-validation. A total of 30 different hyperparameter configurations were compared with RS, resulting in the training and evaluation of 90 models (due to the cross-validation) for 40 epochs each. The possible depth values searched by RS were in the range $[8, 15]$, while the possible widths at each layer were either $64, 320, 576, 832$ or 1088. The depth range was chosen to be as large as possible while still obtaining feasible optimization times. The width range was chosen to cover both relatively small and large widths to easily identify the promising range. The three best models from the RS optimization and their average validations losses on both datasets are presented in Tables 4.5 and 4.6 below.

Depth	Width (nodes per layer)	Val. loss
9	$832, 832, 576, 576, 576, 320, 832, 320, 832$	1.99%
8	$832, 576, 64, 576, 64, 832, 64, 832$	2.01%
15	$320, 832, 576, 64, 1088, 1088, 576, 64, 64, 64, 64, 64, 64, 64, 64$	2.04%

Table 4.5: Depth, width and validation loss for the three best FNNs identified by the first round of RS optimization on the CCP IM dataset.

Depth	Width (nodes per layer)	Val. loss
10	$1088, 64, 320, 320, 320, 576, 1088, 832, 64, 64$	0.77%
10	$1088, 576, 576, 320, 832, 832, 576, 64, 320, 832, 1088, 832, 64, 64$	0.79%
8	$1088, 64, 576, 1088, 320, 320, 320, 320, 64, 832$	0.8%

Table 4.6: Depth, width and average validation loss for the three best FNNs identified by the first round of RS optimization on the SIMM dataset.

The results show that, for both of the datasets, FNNs with depths in the lower range of $[8, 15]$ have lower average validation losses. A second round of optimization was therefore performed using RS again but with smaller depth values and with more possible width values. The depths were now in the range $[2, 9]$, while the widths were in the range $[100, 200, \ldots, 1000]$.

Depth	Width (nodes per layer)	Val. loss
2	600, 600	1.89%
3	800, 300, 500	1.90%
4	600, 500, 500, 900	1.93%

Table 4.7: Depth, width and average validation loss for the three best FNNs identified by the second round of RS optimization on the CCP IM dataset.

Depth	Width (nodes per layer)	Val. loss
2	1000, 1000	0.663%
2	1000, 500	0.677%
9	1000, 400, 100, 100, 100, 100, 100, 100, 100, 100	0.683%

Table 4.8: Depth, width and average validation loss for the three best FNNs identified by the second round of RS optimization on the SIMM dataset.

The results of the second round of RS in Tables 4.7 and 4.8 further show that relatively shallow FNNs seem to have the best performance on both datasets. One notable difference, however, is that all three FNNs in the optimization on the SIMM dataset in Table 4.8 begin with the maximum width of 1000. This suggests that the FNN benefits from having wider layers when trained on the SIMM dataset than when trained on the CCP IM dataset.

Finally, an extensive optimization round of grid search (GS) was performed on the promising hyperparameter ranges identified by RS, again using 3-fold

cross-validation. A total of 150 hyperparameter configurations were thus exhaustively searched for each dataset by training all combinations of FNNs with depths of 2 and 3 and widths of $[200, 400, 600, 800, 1000]$ for the CCP IM dataset and $[100, 500, 1000, 2000, 3000]$ for the SIMM dataset. Tables 4.9 and 4.10 below show the results.

Depth	Width (nodes per layer)	Val. loss
3	$600, 1000, 200$	1.89 %
2	$600, 800$	1.90%
3	$800, 200, 800$	1.91%

Table 4.9: Depth, width and average validation loss for the three best FNNs identified by the GS optimization on the CCP IM dataset.

Depth	Width (nodes per layer)	Test error
3	3000, 1000, 500	0.490%
3	3000, 2000, 500	0.492%
3	3000, 500, 100	0.501%

Table 4.10: Depth, width and average validation loss for the three best FNNs identified by the GS optimization on the SIMM dataset.

The three best-performing FNNs on the SIMM dataset were all found in the last GS optimization (Table 4.10) while the three best models on the CCP IM dataset were found in partly in the GS (Table 4.9) and partly in the second RS optimization (Table 4.8).

4.4.2 Random forest optimization

Before any optimization was performed, a random forest was trained and evaluated with 3-fold cross-validation on both datasets, using the default library hyperparameters in Scikit-learn. The results are shown in Table 4.11.

Dataset / **HP name**	CCP IM	SIMM
n_estimators	100	100
max_features	$\sqrt{m} = \sqrt{286} \approx 16$	$\sqrt{m} = \sqrt{50} \approx 7$
max_depth	∞ (no max depth)	∞ (no max depth)
min_samples_split	2	2
min_samples_leaf	1	1
bootstrap	True	True
Test error:	13.4%	19.0%

Table 4.11: Library default RF hyperparameters (where m is the number of input features), and the test errors of both RF models on the SIMM and CCP IM datasets.

With these results as points of reference, the first round of optimization was performed using RS and 3-fold cross-validation. All hyperparameters were optimized except 'max_features', which was optimized at a later stage using GS. The ranges of possible values for each hyperparameter in the optimization and the results are shown in Table 4.12 below.

Dataset / **HP name**	CCP IM	SIMM
n_estimators	$[20, 40, 60, \mathbf{80}, 100]$	$[20, 40, \mathbf{60}, 80, 100]$
max_depth	$[25, 50, 75, \mathbf{100}, 150, \infty]$	$[25, 50, 75, 100, 150, \boldsymbol{\infty}]$
min_samples_split	$[2, 5, \mathbf{10}]$	$[\mathbf{2}, 5, 10]$
min_samples_leaf	$[\mathbf{1}, 2, 4]$	$[\mathbf{1}, 2, 4]$
bootstrap	[True, **False**]	[True, **False**]
Best test error:	12.9%	18.6%

Table 4.12: Hyperparameter names and corresponding ranges of values searched by RS.

The values in bold gave the best-performing RF model with an average test error of 12.9% and 18.6% for the respective datasets. The next round of optimization with 3-fold cross-validation was done using GS to optimize 'max_features' and further optimize 'n_estimators'. The possible values for 'max_features' were set to approximately $\frac{\sqrt{m}}{2}$, $\sqrt{m}$, $2\sqrt{m}$ and m (where m

is the number of input features of each dataset) and the possible values for 'n_estimators' were set to 50 and 100, while the remaining hyperparameters values were set to the best values previously found by RS. The results of the GS optimization are shown below in Table 4.13,

Dataset / **HP name**	CCP IM	SIMM
n_estimators	[**50**, 100]	[50, **100**]
max_features	[8, 16, **32**, 286]	[3, 7, 25, **50**]
bootstrap	[**True**, False]	[**True**, False]
Best test error:	12.3%	17.3%

Table 4.13: Hyperparameter names and corresponding ranges of values searched by GS.

where the best-performing hyperparameter values are shown in bold. However, these hyperparameter adjustments were unimpressive, only achieving a test error decrease of the respective RFs from 12.9% to 12.3% and from 18.6% to 17.3%.

Chapter 5

Discussion

This section discusses the results obtained in Section 4 and compares them to the results obtained in related work. It also discusses the main limitations of this study and addresses whether the study's research questions could be successfully answered.

5.1 Model performances

5.1.1 Performances of FNNs

The main research question of this study was if an FNN can be used in practice to estimate scenario-based IMs for financial portfolios with relative accuracy. The hypothesis was that FNNs could achieve an error of less than 10%, and the results greatly exceeded this expectation with the best-performing FNNs achieving average estimation errors of 1.5% and 0.49% on the test sets of the CCP IM and SIMM datasets, respectively.

The hyperparameter optimization with RS proved effective in identifying promising intervals for the structural hyperparameters of the FNNs. As shown in Tables 4.7 and 4.8, RS identified that less deep FNNs were better suited for estimating IM on the data in this study, both with SIMM and the CCP IM model. The RS optimization thus allowed for further optimization with GS that would not have been computationally feasible unless the depth of the network was limited to the low depths of 2 and 3. Additionally, the results from Figures 4.1d and 4.2d show that the validation losses of the best-performing models all converge around the training loss, indicating that even non-optimal FNNs seem to avoid overfitting and perform well on unseen portfolios in both datasets.

Efficiency analysis

In addition to the low average validation errors, the FNNs were also found to be almost six times faster at calculating IM than the CCP IM and SIMM models. The execution times to calculate IM for 2500 portfolios (from Table 4.4) are shown again in Table 5.1 together with the respective models' theoretical computational complexities.

IM model	Complexity	n	Constant	Execution time
SIMM	$\mathcal{O}(Cn^2)$	50	C	35.6 s
SIMM FNN	$\mathcal{O}(Wn)$	50	$W = 3\,655\,001$	6.25 s
CCP IM	$\mathcal{O}(nm)$	286	$m = 2500$	54.1 s
CCP IM FNN	$\mathcal{O}(Wn)$	286	$W = 973\,601$	9.40 s

Table 5.1: The theoretical computational complexities and observed execution times of the IM models and their corresponding FNN implementations, where n is the input portfolio size, $m = 2500$ is the number of scenarios used for the CCP IM model, and W is the size of each FNN, measured in its number of trainable parameters.

Comparing the FNN models for SIMM and CCP IM, the observed results are consistent with the theoretical computational complexity for evaluating FNNs. The CCP IM FNN has an approximately 6 times larger portfolio size than the SIMM FNN and approximately 4 times fewer parameters, explaining the $\frac{6}{4} = 1.5 = 50\%$ execution time increase from 6.25 seconds to 9.40 seconds for the two FNN models.

Moreover, comparing the CCP IM model and its FNN implementation, the FNN network size constant W is approximately 390 times larger than the number of scenarios m, yet the FNN is still approximately 6 times faster. This difference implies that CCP IM model implementation is significantly more computationally intensive than the generic scenario-based VaR model with $\mathcal{O}(nm)$ complexity described in Section 2.1.4. Lastly, the SIMM implementation being approximately 6 times slower than its FNN counterpart implies a cost $C \approx 416\,378$ associated with the computation of the parameters mentioned in Section 2.1.3.

It is also worth noting that FNNs' observed efficiency increase is despite the significantly less powerful hardware used to evaluate the FNNs compared to the hardware used with the IM models. Thus, if the limitations of this project (discussed in Section 5.2) do not impede the FNNs' generalization capabili-

ties when used in practice, the assignment provider and other financial institutions should be able to optimize their existing algorithms which currently use scenario-based IM models by replacing these with more efficient FNN models.

5.1.2 Performances of RFs

The secondary research question of this study was if RFs can achieve comparable performances to FNNs, with the purpose of providing a point of reference for the efficacy of FNNs. The results of the study show that RFs perform worse than FNNs, with average estimation errors of 12.3% and 17.3% on test data from the CCP IM and SIMM datasets, compared to the respective FNN errors of 1.5% and 0.49% on the same datasets. This is also consistent with the stated hypothesis that FNNs would outperform RFs by a margin larger than 10%.

Moreover, the hyperparameter optimization for the RF was not sufficiently effective in improving performance to any level comparable to FNNs. The RS and GS optimizations only decreased the average test errors from 13.4% and 19.0% obtained using the default RF hyperparameters, to 12.3% and 17.3% on the respective datasets. These poor performances suggest that RF models are not sufficiently complex to capture either of the IM models. Even as the hyperparameters were adjusted to allow for more complex and less biased models, the average test error never decreased below 12% and 17%, respectively.

5.2 Limitations

5.2.1 Data generation

The main limitation of this project is that the portfolio data generation, described in Section 3.1.2, is not sophisticated enough to accurately represent real portfolios. The $59\ 100$ portfolios in the CCP IM dataset are generated in three different ways, and none of them adequately capture an inherent pattern in the real portfolio data, which is that the DV01 values for a currency pair are typically correlated.

The $20\ 000$ portfolios sampled from the joint normal distribution of the 16 real bank portfolios do not retain the correlation between the DV01 values that originally existed in the individual portfolios from which the joint distribution was created. Another 1600 portfolios were generated by duplicating each of the 16 bank portfolios 100 times and adding uniformly distributed noise to the DV01 values in each portfolio. These portfolios are the most likely to retain the correlation between the DV01 values in the original portfolios, but it is not

guaranteed due to the noise being potentially very large. The remaining $37\,500$ data points were sampled from uniform distributions specifically to diversify the dataset and to avoid obtaining highly correlated portfolios and overfitted models.

The results of quantifying the impact of this data limitation in Section 4.2 show that the best-performing FNN tested on the real provided CCP IM portfolios obtained an average test error of 1.53%. While this is consistent with the FNN's error on the generated test set (1.50%), it does not necessarily indicate that it would obtain similarly low errors when trained on real portfolios. However, the opposite could also be true; the FNN might prove even more effective when trained on real CCP IM portfolios precisely because of the additional patterns in those portfolios that this study failed to capture, but that the FNN can use to improve during training.

This same data limitation does not exist for the SIMM dataset because it only has one sensitivity per currency pair (i.e. between each of the 50 currencies and the dollar), instead of 11 sensitivities per currency pair spread out over a 5-year time horizon, so there is no correlation between sensitivities to be lost in the portfolio generation as there is for CCP IM portfolios. This could partly explain the FNN's stronger performance on the SIMM dataset, along with the SIMM dataset's substantially smaller dimensionality of 50 compared to the CCP IM dataset's dimensionality of 286.

Impact on generalizability

The degree to which the observed FNN performances can generalize to datasets of real portfolios depends heavily on this study's chosen data generation method and the generated portfolios representativeness of real portfolios.

With the benefit of hindsight, this would warrant a thorough examination of both CCP IM and SIMM portfolios to establish an adequate method for generating sufficiently realistic portfolios, as well as a method for evaluating the portfolio generation by comparing the generated portfolios to the real ones. However, this was not considered appropriately at the start of this project, and it was deemed sufficient to evaluate the final FNN models on the real portfolios to assess whether the data generation was sufficiently realistic.

This failed to consider a significant problem which directly impacts the generalization capabilities of the results in this study. Namely, while the obtained FNN errors on the real portfolios were consistent with the errors obtained on the generated test portfolios, this only indicates that the portfolio generation is realistic insofar as that the portfolios are similar to the real ones

used in this study, but not that they would be representative of real portfolios in general. The purpose of the uniform data sampling was precisely to avoid overfitting on the real portfolios, but there was no method to test the assumption that the models does not overfit despite the uniform sampling.

5.2.2 Computational limitations

An additional limitation in this study were the computational resources provided by Google Colab which affected the optimization of the FNN and RF models. The free GPU resources were necessary to obtain feasible model training times, but access to these resources frequently timed out and this drastically decreased the amount of optimization that could be performed within the time frame of the project.

With unrestricted GPU access, the evaluation could be done with more than 3-fold cross-validation, and more possible hyperparameter values could be searched by RS and GS. This would not only allow for finding FNNs with better performance, but the more extensive cross-validation would yield more generalizable results.

5.3 Relation to previous research

This study uses neural networks to estimate IM calculated using both SIMM and a scenario-based CCP IM model. The results of the best-performing model on SIMM data is consistent with the findings of both Villarino and Rodríguez [44] and of Ma et al. [14], who also estimate SIMM IM using deep learning. Ma et al. achieve a 3% mean absolute percentage error when estimating SIMM using an FNN with three hidden layers and with four to eight nodes per layer. This study achieves a lower average error of 0.49% when estimating SIMM using an FNN with three significantly wider layers - consisting of 3000, 2000 and 500 nodes per layer. This performance difference could be explained by the fact that the SIMM portfolios used by Ma et al. are more diverse and contain not only FX instruments, but also other derivatives such as interest rate swaps.

Additionally, the optimal FNN on the CCP IM dataset also has three layers and is also wider than the FNN presented by Ma et al., with 600, 1000 and 200 nodes per layer. It is, however, far less wide than the SIMM FNN obtained in this study. This is interesting because the CCP IM model is more computationally demanding than SIMM (as seen in the observed execution times in Table

5.1), yet it seems to require less complex and wide networks to be approximated optimally. This could be explained by the data generation limitations, as described in Section 5.2, which could yield unrealistic and non-correlated CCP IM portfolios (e.g. due to the uniform sampling) compared to the SIMM portfolio generation where there is not the same issue of correlation between the sensitivities of a currency pair.

Villarino and Rodríguez use a more sophisticated FNN variant called a self-normalizing neural network to estimate IM calculated with SIMM. In their own words, the model showed 'excellent performance' on their dataset, but no measure of the performance on a test set is mentioned in the study. Their model was also relatively small with four hidden layers and 48 nodes per layer, thus further suggesting that SIMM is less complex than the CCP IM model investigated in this study. However, this is not necessarily the case because the smaller size of their network could also be due to their simple dataset portfolios which consist of only interest rate swaps.

Regarding the performance of the RF model, this study could not identify any previous work on using RFs for IM calculations. However, the studies by Probst et al. [51] and Bernard et al. [52] state that library default hyperparameters of RFs are often effective or even near-optimal. This could explain the lack of impact from the hyperparameter optimization on the RF model which only decreased the average test error from 13.4% to 12.3%. However, it could also be explained by the data generation limitations of this study yielding portfolios with non-realistic sensitivity distributions. This is further suggested by the lack of significant impact observed from the extensive hyperparameter optimization that was performed on both FNN models in this study.

5.4 Ethics and sustainability

Unlike traditional algorithms, ML models often behave like black box functions whose output is difficult to understand or reason about. Therefore, ethical considerations when implementing and using ML models are warranted. The purpose of this project is to investigate if an FNN can be used to replace IM models when repeatedly computing portfolio IMs in various algorithms.

Assuming that the FNN is capable of perfectly estimating the IM outputs of a given IM model, whether it is SIMM or a CCP IM model, the only difference would then be that the FNN is significantly more computationally efficient once trained. While the FNNs do not perfectly approximate their corresponding IM models, an average estimation error of 1.5% and 0.49% for the CCP IM model and for SIMM, respectively, is a sufficiently small error to be

negligible in most practical situations.

The results of this study could thereby potentially reduce execution times of algorithms that require repeated IM calculations, such as IM optimization algorithms used for counterparty credit risk reduction, as described in Section 2.1.2. Making such algorithms more efficient would possibly make them more accessible and more widely adopted, thus helping stabilize the financial market and avoiding financial crises by reducing the potential losses of all market participants.

Moreover, increased IM calculation efficiency could also contribute to sustainability to some degree by reducing the overall usage of computational resource of large financial institutions and CCPs. On the other hand, this increased efficiency would also allow said financial institution to increase profits by scaling up operations and performing more calculations than previously, thereby offsetting or even exceeding any potential environmental benefits of the increased efficiency.

Chapter 6

Conclusion

This study presents feedforward neural networks (FNNs) for estimating scenario-based portfolio IM calculated both with a CCP IM model and with SIMM. The optimal FNNs were found using the hyperparameter optimization algorithms random search (RS) and grid search (GS) with 3-fold cross-validation. The two final optimized FNNs for estimating the CCP IM model and SIMM both consist of three hidden layers with the former having 600, 1000 and 200 nodes per layer, and the latter having 3000, 2000 and 500. They achieve strong performances with mean absolute percentage errors (MAPEs) of 1.5% and 0.49% on the generated CCP IM and SIMM datasets in this study, and they calculate IM approximately six times faster than their IM model counterparts. In theory, trained FNNs could thus be used to replace significantly less efficient IM models to optimize algorithms that require multiple IM calculations. However, due to the limitations in how the training dataset was generated, the results of this study are not sufficient to make a general assertion on the efficacy of FNNs for estimating scenario-based IM because they might achieve worse performance on real portfolio data.

The study also implemented and optimized two RF models, both of which achieved significantly worse test errors of 12.3% and 17.3% on the CCP IM and SIMM datasets, respectively. This could imply that RFs are insufficiently complex and poorly suited for the learning problem of estimating IM, but this cannot be concluded because of the previously mentioned limitations in the dataset generation.

6.1 Further work

This study indicates that FNNs could potentially be suitable models for scenario-based portfolio IM estimation. However, in order to draw meaningful conclusions about the efficacy of FNNs for this learning problem, further research is warranted. The primary question raised by this study is whether the results can be generalized to datasets of real portfolios, and addressing this question requires access to a large number of financial portfolios or to more sophisticated methods of generating such portfolios. This study also did not have direct access to the IM models that were used to compute the portfolio IMs and could therefore not properly compare the execution times of the FNNs and the IM models using the same hardware. To assess the impact of the dataset generation limitations in this study and the overall generalizability of the results, an interesting avenue that could be pursed in the context of computing dynamic IM would be to replace the use of SIMM in the work of Zeron and Ruiz [13] with FNNs trained using the sensitivities obtained from Chebyshev tensors.

Bibliography

[1] Michael I Jordan and Tom M Mitchell. "Machine learning: Trends, perspectives, and prospects". In: *Science* 349.6245 (2015), pp. 255–260.

[2] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. "Automated machine learning: State-of-the-art and open challenges". In: *arXiv preprint arXiv:1906.02287* (2019).

[3] Oludare Isaac Abiodun et al. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon* 4.11 (2018).

[4] Adam Fadlalla and Chien-Hua Lin. "An analysis of the applications of neural networks in finance". In: *Interfaces* 31.4 (2001), pp. 112–122.

[5] Ismael Alexander Boudiaf, Martin Scheicher, and Francesco Vacirca. "CCP initial margin models in Europe". In: *ECB Occasional Paper* 2023/314 (2023).

[6] Will Kenter. *Financial Instruments Explained: Types and Asset Classes*. Accessed on April 13, 2023. Feb. 2023. URL: https://www.investopedia.com/terms/f/financialinstrument.asp.

[7] *International Accounting Standard (IAS) 32.11*. Accessed on April 13, 2023. URL: https://www.iasplus.com/en/standards/ias/ias32.

[8] James Chen. *What Is Forex Trading? A Beginner's Guide*. Accessed on April 15, 2023. Apr. 2023. URL: https://www.investopedia.com/articles/forex/11/why-trade-forex.asp.

[9] James Chen. *What Is Spot Trading and How Do You Profit? How It Works*. Accessed on May 5, 2023. May 2022. URL: https://www.investopedia.com/terms/s/spottrade.asp.

[10] Rajeev Dhir. *Forward Contract: How to Use It, Risks, and Example*. Accessed on May 5, 2023. July 2022. URL: https://www.investopedia.com/terms/f/forwardcontract.asp.

[11] International Swaps and Inc Derivatives Association. *ISDA SIMM Methodology, version 2.6*. Accessed on 2023-12-28. September 15, 2023. URL: https://www.isda.org/2023/09/15/isda-publishes-isda-simm-methodology-version-2-6/.

[12] Basel Committee on Banking Supervision. *Margin requirements for non-centrally cleared derivatives*. Accessed on 2024-04-28. 2020. URL: https://www.bis.org/bcbs/publ/d499.pdf.

[13] Ignacio Ruiz and Mariano Zeron. "Dynamic Initial Margin via Chebyshev Tensors". In: *arXiv preprint arXiv:1808.08221* (2018).

[14] Xun Ma et al. "Initial margin simulation with deep learning". In: *Available at SSRN 3357626* (2019).

[15] Reserve Bank of Australia. "Central counterparty margin frameworks". In: *RBA Bulletin* December (2017), pp. 79–88.

[16] European Union Emissions Trading System. *Margin period of risk ('MPOR')*. Accessed on 2024-05-11. 2015. URL: https://emissions-euets.com/internal-electricity-market-glossary/853-margin-period-of-risk-mpor.

[17] Bloomberg. *The ISDA SIMM Overview FAQ*. Accessed on 2023-12-28. 2019. URL: https://data.bloomberglp.com/professional/sites/10/The-ISDA-SIMM-Overview-FAQ.pdf.

[18] Anders Krogh. "What are artificial neural networks?" In: *Nature biotechnology* 26.2 (2008), pp. 195–197. Graphics are used with permission from copyright holder (Springer Nature).

[19] Mansfield Merriman. *A List of Writings Relating to the Method of Least Squares: With Historical and Critical Notes*. Vol. 4. Academy, 1877.

[20] Bayya Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[21] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[22] M. Minsky and S. Papert. *Perceptrons*. Cambridge, MA: MIT Press, 1969.

[23] Paul J. Werbos. “Applications of advances in nonlinear sensitivity analysis”. In: *System Modeling and Optimization*. Ed. by R. F. Drenick and F. Kozin. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 762–770. ISBN: 978-3-540-39459-4.

[24] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016, p. 173. ISBN: 978-0262035613.

[26] Murat H Sazlı. “A brief review of feed-forward neural networks”. In: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* 50.01 (2006).

[27] Daniel Berrar. “Cross-Validation”. In: *Encyclopedia of Bioinformatics and Computational Biology*. Ed. by Shoba Ranganathan et al. Oxford: Academic Press, 2019, pp. 542–545. ISBN: 978-0-12-811432-2.

[28] Mengting Niu et al. “RFAmyloid: A Web Server for Predicting Amyloid Proteins”. In: *International journal of molecular sciences* 19 (July 2018). DOI: 10.3390/ijms19072071. Source uses CC license.

[29] Gareth James et al. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN: 1461471370.

[30] Jane Elith, John R Leathwick, and Trevor Hastie. “A working guide to boosted regression trees”. In: *Journal of animal ecology* 77.4 (2008), pp. 802–813.

[31] Scikit-learn. *Decision Tree Regression*. Accessed on May 4, 2023. n.d. URL: https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html. Source uses BSD license.

[32] Trevor Hastie et al. “Random forests”. In: *The elements of statistical learning: Data mining, inference, and prediction* (2009), pp. 587–604.

[33] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. “An Efficient Approach for Assessing Hyperparameter Importance”. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Beijng, China: PMLR, June 2014, pp. 754–762.

[34] Jan N Van Rijn and Frank Hutter. "Hyperparameter importance across datasets". In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 2367–2376.

[35] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2 (2012).

[36] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[37] Shiliang Sun et al. "A survey of optimization methods from a machine learning perspective". In: *IEEE transactions on cybernetics* 50.8 (2019), pp. 3668–3681.

[38] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[40] Sunitha Basodi et al. "Gradient amplification: An efficient way to train deep neural networks". In: *Big Data Mining and Analytics* 3.3 (2020), pp. 196–207. DOI: 10.26599/BDMA.2020.9020004.

[41] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE].

[42] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV].

[43] Scikit-learn. *Random Forest Regression*. Accessed on May 4, 2023. n.d. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

[44] Joel Pérez Villarino and Álvaro Leitao Rodríguez. "Deep Learning-Based Method for Computing Initial Margin". In: *Engineering Proceedings* 7.1 (2021), p. 41.

[45] Justin Chan, Shengyao Zhu, and Boris Tourtzevitch. "Practical Approximation Approaches to Forecasting and Backtesting Initial Margin Requirements". In: *Available at SSRN 3079782* (2017).

[46] François Chollet et al. *Keras*. Accessed May 8, 2023. 2015. URL: https://keras.io/.

[47] James O'Malley et al. *Keras Tuner*. Accessed May 8, 2023. 2020. URL: https://keras-team.github.io/keras-tuner/.

[48] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[49] Google. *Google Colaboratory*. Accessed on May 08, 2023. URL: https://colab.research.google.com.

[50] JetBrains. *PyCharm IDE*. Accessed on May 09, 2023. 2021. URL: https://www.jetbrains.com/pycharm/.

[51] Philipp Probst, Marvin N Wright, and Anne-Laure Boulesteix. "Hyperparameters and tuning strategies for random forest". In: *Wiley Interdisciplinary Reviews: data mining and knowledge discovery* 9.3 (2019), e1301.

[52] Simon Bernard, Laurent Heutte, and Sébastien Adam. "Influence of Hyperparameters on Random Forest Accuracy". In: *Multiple Classifier Systems*. Ed. by Jón Atli Benediktsson, Josef Kittler, and Fabio Roli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 171–180. ISBN: 978-3-642-02326-2.

TRITA-EECS-EX-2025:90
Stockholm, Sweden 2024

www.kth.se