UTRECHT UNIVERSITY

DEPARTMENT OF MATHEMATICS

MASTER THESIS

Implementing Deep Neural Networks to Improve the Financial Market Model in the Uniform Calculation Method

Author:
Tianyi Chen
5670500

Supervisors:
Dr. K. Dajani
Drs. H. J. M. de Bock
C. Dekker

Second reader:
Dr. M. C. J. Bootsma

Utrecht University

RiskCo
FINANCIAL MODEL SOLUTIONS

July 5, 2019

Abstract

In the Netherlands, the pension system is a large and highly-developed financial sector. However, it remains a challenging and serious issue for pension funds to provide accurate pension forecasting and sufficient pension communication, and the consequence of a wrong estimate of pension entitlements can be destructive for participants. To empower the pension participants to obtain insights in expected incomes and get engaged in active retirement planning, the Uniform Calculation Method (Uniforme Rekenmethodiek, URM) was issued by DNB in 2015 and has come in force since the beginning of 2019. All the pension service providers in the Netherlands are required to implement URM to estimate the pension entitlements so as to determine the risk attitude and provide the participants with accurate pension communication.

In this thesis, the overall goal is to investigate if deep neural networks can be constructed and implemented to improve the financial market model in URM so as to help provide better financial forecasting and pension planning. Thus, both the financial market model in URM and a set of DNNs are developed to generate forecasting scenarios. Accordingly, the pension entitlement development curves are estimated, and a comparison as well as an evaluation can be conducted to answer the research question. Based on the chosen experimental settings and the considered training, validation and test periods, the conclusion is that the DNN models implemented in this thesis can outperform the URM model and provide relatively more accurate pension communication, even though the accuracy still needs to be further improved.

Acknowledgements

This research project is conducted in the form of an internship at the Dutch company RiskCo B.V. in Utrecht as the final requirement to obtain the Master's degree in Mathematics at Utrecht University.

First of all, I would like to thank my UU supervisor Karma Dajani, who referred me to RiskCo, supervised my work, helped me whenever I had problems and encouraged me during the whole thesis project. Professor Karma is also the supervisor of my bachelor thesis, and I am grateful for the time, support, patience and care she has devoted to my study for all these years.

Then, I would like to thank my RiskCo supervisor Bert de Bock for not only offering me this valuable opportunity to write my thesis during the internship at RiskCo, but also supporting me, supervising me, giving me professional feedback and helping me practice multiple presentations in every stage of the project.

I would also like to specifically thank the other RiskCo supervisor Connor Dekker, who was always there and was always patient when I needed help. He offered me immediate feedback and detailed advice during all these months. Without his help I would not have been able to finish this thesis project.

Further, I would like to thank my colleagues at RiskCo for all the assistance and care. It has been a great memory to work there and I have gained not only experience but also friendship.

Finally, I would like to show my appreciation to my family, friends, boyfriend as well as all the extraordinary classmates and professors that I have met, without whom I could not have made it this far in my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This research thesis project is conducted in the form of an internship at RiskCo B.V. in Utrecht, the Netherlands. This chapter starts with a general introduction to the company. Then, the relevant background is briefly described, and the research questions are addressed. Accordingly, an overview of the methodology is provided without detail to help capture the entire perspective of the research. In the end, the outline of the thesis report is presented.

1.1 RiskCo B.V.

RiskCo B.V., founded in 2002, is a consultancy firm working on the intersection of business, actuary and IT in the financial world. The company started as the distributor of ProductXpress, a workbench for financial product development and calculations, and extended his offerings in, among others, the areas of data quality and reporting for regulators.

The company can be characterized by:

- Performing projects for Life Insurance companies and Pension Funds, for the liabilities and assets parts of the Balance sheet.
- Strategic partner of DXC for the implementation of the DXC ProductXpress calculation tool, their most important IT platform for large scale calculations.
- 120 academics trained in mathematics, finance, business studies, economics, econometrics, physics, software development and artificial intelligence.
- Offices in the Netherlands, Portugal and the Philippines.
- Performed projects in 16 jurisdictions in Europe, Asia, Australia, Africa & the Americas.
- Part of Praxis IFM.

Examples of projects are:

- Development and implementation of a platform for reporting the liabilities to DNB, for a Dutch system administrator for pension funds;
- Reserve calculations and capital requirements calculations under Solvency II (Redesign of IT landscape for a Dutch insurance company);
- Implementation of product calculation engines for new or existing administration systems, quotation systems and financial planners;

- Investment rule optimizations for banks; using replicating portfolios;
- Multiple data quality audits, using the RiskCo methodology on pension fund administrations;
- Implementation of the rules and calculations for illustration and administration purposes of hundreds of products across many lines of business for a very large international insurance company.

In the Netherlands RiskCo worked among other things for ASR, A&O services, Klaverblad, PGGM, Nationale Nederlanden, Vivat and Delta Lloyd. Worldwide the US based insurance company METLIFE is a customer. In 2017 and 2018, Aon Hewitt Benefits Administration and InAdmin NV from APG were taken over by RiskCo.

1.2 Background

In the Netherlands, the pension system is a large and highly-developed financial sector. According to the OECD report in 2018, the pension assets in the Netherlands worth 184% of GDP, which leads the Netherlands to rank the second among all of the 36 OECD countries [17]. There are three pillars in the Dutch pension system, namely, state pension, collective pension schemes and individual pension products. More than 90% of the Dutch citizens participate in the collective pension schemes which are administrated by pension fund service providers [20]. Thus, the pension accrual from the second pillar, collective pension schemes, is an important component of the participants' retirement income.

The consequence of a wrong estimate of the pension entitlements from the second pillar can be destructive. The participants may not be able to conduct effective retirement planning before the retirement and may be confronted with unexpected financial dilemmas after the retirement. Therefore, accurate pension forecasting and sufficient pension communication are necessary.

To empower the pension participants to obtain insights in expected incomes and get engaged in active retirement planning, the Uniform Calculation Method (Uniforme Rekenmethodiek, URM) was issued by DNB in 2015 and has come in force since the beginning of 2019 [2]. All the pension service providers in the Netherlands are required to implement URM to estimate the pension entitlements so as to determine the risk attitude and provide the participants with accurate pension communication.

1.3 Research Questions

The overall goal of this thesis is to investigate if deep neural networks can be constructed and implemented to improve the financial market model in URM so as to help provide better financial forecasting and pension planning.

1.4 Overview of Methodology

In this thesis we focus on constructing and refining deep neural networks (DNNs), which can consist of convolutional layers and recurrent layers. We build the baseline model, the URM financial market model and the deep learning neural network in

Python. Then, together with assumptions such as investment strategies and life expectancy for a virtual participant, we can forecast the individual pension entitlement development using each of the model and then do backtest. That is, after getting the real-world pension entitlement development based on the real-world data, we can compute and compare the mean squared errors between the real-world data and the forecasts generated by the different models, and we can reach the conclusion if our model can outperform the baseline model as well as URM model. The naive method is used as the baseline model to check the triviality of URM model and deep learning model.

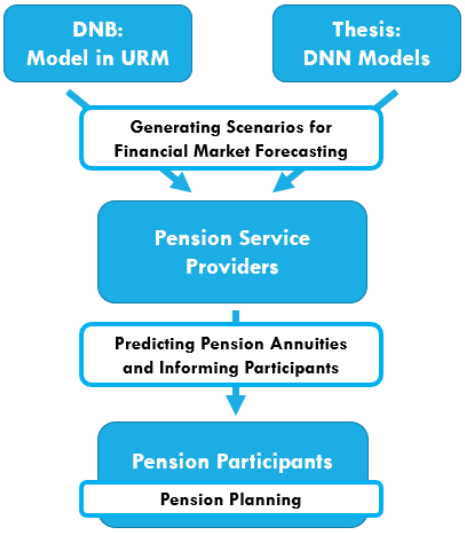


FIGURE 1.1: Illustration of the URM workflow.

1.5 Outline

This thesis consists of three parts.

The first part is devoted to background introduction and literature overview. In chapter 2, the Dutch pension system, relevant context and specifically URM are briefly introduced. In chapter 3, the research question is framed in detail. Chapter 4 is about an overview of approaches to time series forecasting including the classical methods and the technique of deep neural networks.

The second part is devoted to the introduction to URM. In chapter 5, the mathematical foundations for stochastic calculus is given for the later introduction to the financial market model in URM. In chapter 6, the regulations and financial market model in URM are elaborated.

The third part is mainly about the deep learning model and the methodology of the research project. In chapter 7, the foundations for deep learning is presented. In chapter 8, the methodology of the research and the architecture of the deep learning model are elaborated. In chapter 9, the results including the evaluations of all concerned models are given. In chapter 10, the conclusion of this thesis is presented. Last but not least, chapter 11 is devoted to further discussion and future work.

Chapter 2

Dutch Pension System and URM

In this chapter, the Dutch pension system, relevant context and specifically URM are briefly introduced. The main references in this chapter include an overview article *The Dutch Pension System* issued by the the Dutch Association of Industry-wide Pension Funds (VB) [20] and the webpage article *Rekenmethodieken voor weergave van ouderdomspensioen in scenario's* by the Dutch Central Bank (DNB) [2].

2.1 Dutch Pension System

2.1.1 Three-Pillar System

The Dutch pension system consists of three main pillars, namely, the state pension (AOW), the collective pension schemes and the individual pension products.

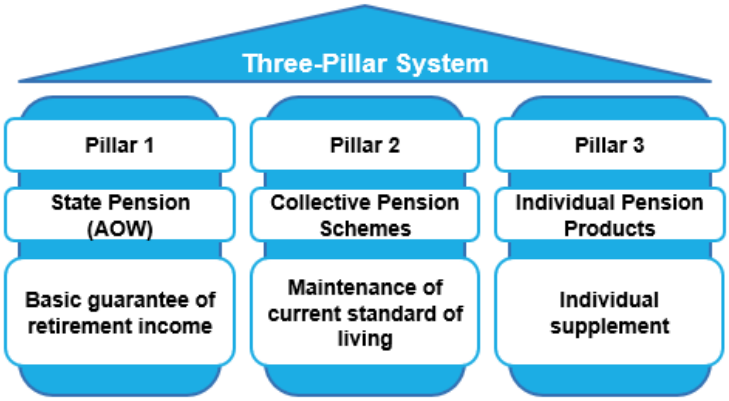


FIGURE 2.1: The three pillars in the Dutch pension system.

State Pension (AOW)

The first pillar in the Dutch pension system is the state pension, which is the so-called General Old Age Pension Act (Algemene Ouderdomswet, AOW). The state pension provides a basic income which is related to the statutory minimum wage for the Dutch residences who reach the legal retirement age [1]. The state pension is mostly funded by contributions from the workforce in the Netherlands, which is the so-called pay-as-you-go system. Additional funding for the state pension is from government public funds. The first pillar of the Dutch pension system provides the residences with the basic guarantee of retirement income.

[1] The retirement age is not constant and is currently 66 years and 4 months [23]

Collective Pension Schemes

The second pillar in the Dutch pension system is the collective pension schemes, which is the focus of this thesis. In the Netherlands, more than 90% of the residents participate in the collective pension schemes and accrue additional retirement incomes [26]. These pension schemes are mostly managed by occupational or company-based pension funds but also in some cases by insurance companies. If required by the industry or the company where the participants work for, then it is mandatory for the participants to contribute a certain amount of premium monthly or annually. After retirement, the participants can receive pension entitlements depending on the pension agreements made with the pension providers.

Individual Pension Products

The third pillar in the Dutch pension system is the individual pension products, which is mainly used as a substitution or supplement of the second pillar. Self-employed residents or employees who do not participate in the collective pension schemes usually purchase products in the third pillar. Personal investments and savings are also a part of the third pillar.

2.1.2 Pension Schemes

The two most important pension schemes in the Netherlands are Defined Benefits (DB) schemes and Defined Contribution (DC) schemes. In DB schemes, the pension benefits or entitlements are predefined, whereas the pension contribution or premium varies. In DC system, the contribution is specified, but the benefit can vary depending on the result of the pension providers' investments. In this thesis we focus on the DC scheme to see how the entitlement can be estimated based on the financial market forecast.

2.1.3 Pension Regulation and Communication

In the Netherlands, there are two regulators in charge of supervising pension providers, namely, the Dutch Central Bank (de Nederlandsche Bank, DNB) and the Dutch Authority for the Financial Markets (de Autoriteit Financiële Markten, AFM). DNB is responsible for examining if the financial position of the pension funds is healthy and sustainable according to the new Financial Assessment Framework (nieuwe Financieel Toetsingskader, nFTK) and other regulations. AFM is responsible for monitoring if the pension funds are behaving legally and performing normally as well as if the investment supports and information are provided to the participants accurately and timely according to pension communication regulations.

Pension communication is essential for retirement planning. Participants should be informed of an accurate expected retirement income from the second pillar to see if they need to make supplements via the third pillar in order to cover the expected expenses after retirement. Various types of communication materials, such as letters, emails and brochures, can be offered by the pension providers to inform the participants of the retirement accrual [26]. As shown in figure 2.2, some navigation metaphors are applied to help better illustrate the information.

FIGURE 2.2: An example of the navigation metaphors used in communication materials [26].

Insufficient or inaccurate pension communication can lead to incorrect retirement planning and cause unpleasant financial dilemmas and unexpected drops in the life quality at the time of retirement. To empower the pension participants to obtain insights in expected incomes and get engaged in active retirement planning, the Uniform Calculation Method (Uniforme Rekenmethodiek, URM) has come in force in the pension forecast and communication since the beginning of 2019, which is briefly introduced in the next section.

2.2 Uniform Calculation Method (URM)

As indicated in the name, the Uniform Calculation Method is a methodology to estimate the future pension entitlements and to generate uniform scenario sets that can be implemented by all the pension administrators in the Netherlands. URM was issued by DNB in 2015 and has become mandatory for all the pension providers since the beginning of 2019 [2]. The goal of URM is to enhance the pension communication and to accurately inform the participants the amount of pension benefits they are expected to receive after their retirement. Besides, URM can also assist the pension providers to determine the risk attitude.

A scenario set including 2,000 scenarios for 60 projection years is published by DNB every quarter. It is regulated that three scenarios have to be taken into consideration, namely, the 50^{th} percentile as the expected scenario, the 95^{th} percentile as the optimistic scenario and the 5^{th} percentile as the pessimistic scenario. Based on the URM scenario set and certain assumptions such as a certain starting value of the pension fund and a specific investment strategy, a pension entitlement development curve can be obtained to illustrate the changes of an individual participant's annual entitlement during his or her lifetime, as shown in the concept figure 2.3.

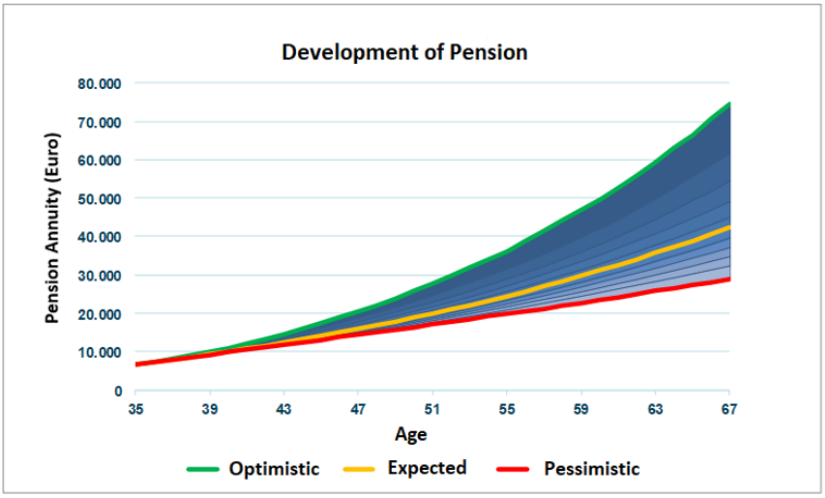


FIGURE 2.3: The pension entitlement development curve expected to be generated with the URM scenario set [19].

There are many aspects in URM, including the premium policy, investment policy, financial market, changes in pension scheme, demographic assumptions, etc. However, the lion's share of the mathematics-related part in URM is about forecasting the financial market, including estimating the future interest rates, bond returns and stock returns, which is a time series forecasting model. For many other aspects such as asset strategies, life expectancy and so forth, some regulations or laws that are not closely related to mathematics are issued and assumptions are directly offered. In this thesis, we focus on the financial market model in URM, specifically, the nominal interest rates, nominal bond returns and nominal stock returns. Because the construction and forecast of inflation rates can be a wide research topic that is conducted by another separate research project at RiskCo B.V., in this thesis project only the evaluation towards the nominal terms are considered.

Chapter 3

Overview of Approaches to Time Series Forecasting

A time series refers to a sequence of data points ordered in discrete or continuous time. In data analysis, time series forecasting is one of the most important topics, which is to apply the previously observed data to predict the future data. Financial market forecasting, the core of this thesis, is a typical time series forecasting problem. In this chapter, we briefly overview various approaches to time series forecasting.

3.1 Classical Method

3.1.1 Persistence Forecasting Models

In the persistence forecasting models, the data points in the past are directly used as the forecast for the future. This kind of models are considered to be the simplest and are conventionally used as a baseline model. Any model that cannot perform better than the persistence forecasting models is regarded as trivial. In this thesis, the baseline model is a persistence forecasting model that uses the last previously observed data point as the forecast.

3.1.2 Stochastic Differential Equations (SDE)

The stochastic differential equations (SDE) are differential equations incorporating one or more stochastic terms, which is well established and widely implemented in scientific research. The financial market model in URM is developed intensively based on the theory in stochastic calculus, which is elaborated in chapter 4 and 5.

3.1.3 Vector Autoregressive (VAR) Models

The vector autoregressive (VAR) model is one of the most popular approaches for generating macroeconomics scenarios in industry. A VAR model expresses a stochastic process in terms of its delayed version. It describes the correlation between variables as well as the autocorrelation, which is the correlation between variables through time. A p^{th}-order VAR model refers to the VAR model with p lags in time, and it can be formulated as

$$Y_t = \alpha + \sum_{i=1}^{p} \Gamma_i Y_{t-i} + e_t,$$

where Y_t is a n-vector for n variables, α is a constant n-vector of offsets, Γ_i is the $n \times n$ autoregressive matrix and e_t is the n-vector of errors that satisfies certain conditions [24].

3.2 Deep Neural Network

All the aforementioned models have been intensively applied and well studied. However, in the recent few years, attention has been focused on and breakthroughs have been continuously made in deep learning, a relatively new field. Even though there are already certain mathematical foundations, various powerful algorithms and many advanced publications from research institutes and companies like Google and Uber, more companies like investment banks or trading firms might have chosen to keep their researches confidential, so not many open source deep learning models that can be directly applied in industry are available. Thus, considering the great potential of deep learning, the relatively large possibility for innovation in this field and the current competition regarding applying AI techniques, it is promising to apply deep learning to improve URM. The theory of deep learning is elaborated in chapter 6.

Chapter 4

Foundation for Stochastic Calculus

In this chapter, the mathematical foundation for stochastic calculus is given for the later introduction to the financial market model in URM. The main reference of this chapter is *Stochastic Calculus for Finance II* by Steven E. Shreve [21].

4.1 Brownian Motion

Definition 4.1.1 (Brownian motion). *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. Given $t \geq 0$, assume for each $\omega \in \Omega$ there exists a continuous function $W(t)$ that depends on ω with initial value $W(0) = 0$. Then $W(t)$ of $t \geq 0$ is a Brownian motion, if for all time partitions $0 = t_0 < t_1 < ... < t_m$ it is satisfied by all the increments*

$$W(t_{i+1}) - W(t_i), \forall i = 0, 1, ..., m-1$$

that (i) all the increments are independent
and (ii) each of the increment has the normal distribution $N(0, t_{i+1} - t_i)$.

Definition 4.1.2 (Filtration for Brownian motion). *Assume $W(t)$ for $t \geq 0$ is a Brownian motion defined on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Then a collection of σ-algebras $\mathcal{F}(t)$ for $t \geq 0$ is a filtration for $W(t)$ if the following properties are all satisfied:*
(i) for $0 \leq s < t$, $\mathcal{F}(s) \subseteq \mathcal{F}(t)$;
(ii) $\forall t \geq 0$, $W(t)$ is $\mathcal{F}(t)$-measurable;
(iii) for $0 \leq s < t$, $W(t) - W(s)$ is independent of $\mathcal{F}(s)$.

Theorem 4.1.1. *Brownian motion is a martingale.*

Proof. For $0 \leq s \leq t$,

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

According to definition 4.1.1, $W(t) - W(s)$ is independent of $\mathcal{F}(s)$, and $W(s)$ is $\mathcal{F}(s)$-measurable, so we have

$$\mathbb{E}[W(t) - W(s)|\mathcal{F}(s)] = \mathbb{E}[W(t) - W(s)] = 0,$$

$$\mathbb{E}[W(s)|\mathcal{F}(s)] = W(s).$$

Thus,

$$\mathbb{E}[W(t)|\mathcal{F}(s)] = W(s).$$

□

By definition, $W(0) = 0$, which is the initial information we know for sure, so we get

$$\mathbb{E}[W(t)] = \mathbb{E}[W(t)|\mathcal{F}(0)] = W(0) = 0.$$

Definition 4.1.3 (Quadratic variation). *Assume function $f(t)$ is defined for $0 \leq t \leq T$. Then the quadratic variation of f up to T is*

$$[f,f](T) = \lim_{||\Pi||\to 0} \sum_{j=0}^{n-1} [f(t_{j+1}) - f(t_j)]^2,$$

where $\Pi = \{t_0, t_1, ..., t_n\}$ is the time partition with $0 < t_0 < t_1 < ... < t_n = T$ and $||\Pi|| = \max_{j=0,...,n-1}(t_{j+1} - t_j)$.

Theorem 4.1.2. *If $W(t)$ is a Brownian motion, then its quadratic variation up to time T is $[W,W](T) = T, \forall T \geq 0$, with L^2-convergence.*

Proof. Taking a partition $\Pi = \{t_0 = 0, t_1, ..., t_n = T\}$ of $[0,T]$, we can define

$$Q_\Pi = \sum_{j=0}^{n-1} (W(t_{j+1}) - W(t_j))^2,$$

which is a random variable that depends on the realization of the Brownian motion path. By definition, we can denote the quadratic variation as

$$[W,W](T) = \lim_{||\Pi||\to 0} Q(\Pi).$$

Now our goal is to show that as $||\Pi|| \to 0$, the expectation of Q_Π is T, and the variance of Q_Π vanishes.
By definition,

$$\begin{aligned}
\mathbb{E}[(W(t_{j+1}) - W(t_j))^2] &= Var[W(t_{j+1}) - W(t_j)] + \mathbb{E}[W(t_{j+1}) - W(t_j)]^2 \\
&= t_{j+1} - t_j + 0 \\
&= t_{j+1} - t_j.
\end{aligned}$$

Then, we obtain the expectation of Q_Π,

$$\mathbb{E}Q_\Pi = \sum_{j=0}^{n-1} \mathbb{E}[(W(t_{j+1}) - W(t_j))^2] = \sum_{j=0}^{n-1} (t_{j+1} - t_j) = T.$$

To get the variance of Q_Π, we have

$$\begin{aligned}
Var[(W(t_{j+1}) - W(t_j))^2] &= \mathbb{E}\Big[\Big((W(t_{j+1}) - W(t_j))^2 - \mathbb{E}[(W(t_{j+1}) - W(t_j))^2]\Big)^2\Big] \\
&= \mathbb{E}\Big[\Big((W(t_{j+1}) - W(t_j))^2 - (t_{j+1} - t_j)\Big)^2\Big] \\
&= \mathbb{E}\Big[(W(t_{j+1}) - W(t_j))^4\Big] - 2(t_{j+1} - t_j)\mathbb{E}\Big[(W(t_{j+1}) - W(t_j))^2\Big] \\
&+ (t_{j+1} - t_j)^2 \\
&= \mathbb{E}\Big[(W(t_{j+1}) - W(t_j))^4\Big] - 2(t_{j+1} - t_j)(t_{j+1} - t_j) + (t_{j+1} - t_j)^2 \\
&= \mathbb{E}\Big[(W(t_{j+1}) - W(t_j))^4\Big] - (t_{j+1} - t_j)^2
\end{aligned}$$

By definition, $W(t_{j+1}) - W(t_j) \sim N(0, t_{j+1} - t_j)$, so we can directly know that its fourth moment is

$$\mathbb{E}\Big[\Big(W(t_{j+1}) - W(t_j)\Big)^4\Big] = 3(t_{j+1} - t_j)^2.$$

Thus, we get

$$Var[(W(t_{j+1}) - W(t_j))^2] = 3(t_{j+1} - t_j)^2 - (t_{j+1} - t_j)^2 = 2(t_{j+1} - t_j)^2$$

Then we can obtain the variance of Q_Π,

$$\begin{aligned} Var(Q_\Pi) &= \sum_{j=0}^{n-1} Var[(W(t_{j+1}) - W(t_j))^2] = \sum_{j=0}^{n-1} 2(t_{j+1} - t_j)^2 \\ &\leq \sum_{j=0}^{n-1} 2||\Pi||(t_{j+1} - t_j) = 2||\Pi||T. \end{aligned}$$

Thus, as $||\Pi|| \to 0$ we have $Var(Q_\Pi)$ converging to 0, which leads to

$$[W, W](T) = T. \tag{4.1}$$

□

According to Theorem 4.1.2 we know that up to some time T_1 the quadratic variation of the Brownian motion is T_1, and up to some time T_2 such that $0 < T_1 < T_2$, the quadratic variation is T_2, so the quadratic variation accumulated during $[T_1, T_2]$ is $T_2 - T_1$. Thus, we can get the conclusion that "Brownian motion accumulates quadratic variation at rate one per unit time" [21].
Equation 4.1 can be written in the differential form as

$$dW(t)dW(t) = dt.$$

To extend our differential multiplication table, we also want to evaluate $dW(t)dt$, $dtdW(t)$ and $dtdt$. Taking a partition $\Pi = \{t_0 = 0, t_1, ..., t_n\}$, then we can denote the maximum step size as $||\Pi|| = \max_{j=0,...,n-1}(t_{j+1} - t_j)$. Firstly, we have

$$\begin{aligned} &\lim_{||\Pi||\to 0} \Big| \sum_{j=0}^{n-1} (t_{j+1} - t_j)(W(t_{j+1}) - W(t_j)) \Big| \\ &\leq \lim_{||\Pi||\to 0} \max |W(t_{j+1}) - W(t_j)| \sum_{j=0}^{n-1} (t_{j+1} - t_j) \\ &= \lim_{||\Pi||\to 0} \max |W(t_{j+1}) - W(t_j)| \cdot T \\ &= 0 \cdot T = 0. \end{aligned} \tag{4.2}$$

Thus, we have

$$\lim_{||\Pi||\to 0} \sum_{j=0}^{n-1} (t_{j+1} - t_j)(W(t_{j+1}) - W(t_j)) = 0,$$

which can be written in the differential form as $dW(t)dt = dtdW(t) = 0$.

Besides, we have

$$\begin{aligned}
&\lim_{||\Pi||\to 0} \sum_{j=0}^{n-1} (t_{j+1} - t_j)^2 \\
&\leq \lim_{||\Pi||\to 0} \max |t_{j+1} - t_j| \sum_{j=0}^{n-1} (t_{j+1} - t_j) \\
&= \lim_{||\Pi||\to 0} ||\Pi|| \cdot T = 0,
\end{aligned} \tag{4.3}$$

which can be written in the differential form as $dtdt = 0$.
In conclusion, we have the following differential multiplication table:

$$dW(t)dW(t) = dt, \qquad dW(t)dt = dtdW(t) = 0, \qquad dtdt = 0. \tag{4.4}$$

4.2 Stochastic Calculus

Definition 4.2.1 (Itô's integral). *Let $\{W(t) : t \geq 0\}$ be a Brownian motion with a filtration $\{\mathcal{F}(t) : t \geq 0\}$. Let $\{\Delta(t) : t \geq 0\}$ be an adapted stochastic process. Then Itô's integral is defined as*

$$I(t) = \int_0^t \Delta(s) dW(s).$$

We first define Itô's integrals for simple integrands $\Delta(t)$, and then extend the results to the ones with general integrands.
Taking a partition $\Pi = \{t_0 = 0, t_1, ..., t_n = T\}$ of $[0, T]$, now we consider a simple process $\{\Delta(t) : 0 \leq t \leq T\}$ which is constant on each subinterval $[t_j, t_{j+1})$.
In this case, if $t_k \leq t \leq t_{k+1}$, the Itô's integral can be rewritten as

$$I(t) = \sum_{j=0}^{k-1} \Delta(t_j)[W(t_{j+1}) - W(t_j)] + \Delta(t_k)[W(t) - W(t_k)], \tag{4.5}$$

which refers to the Itô's integral with simple integrands.

Theorem 4.2.1. *The Itô's integral with simple integrands, as defined by equation 4.5, is a martingale.*

Proof. Given $0 \leq s \leq t \leq T$, our goal is to show $\mathbb{E}[I(t)|\mathcal{F}(s)] = I(s)$. Let Π be a partition of $[0, T]$. Assume $s \in [t_l, t_{l+1})$ and $t \in [t_k, t_{k+1})$ with $t_l < t_k$, so that s and t are in different subintervals of Π. If s and t are in the same subinterval, the proof should be simpler, so here we only consider the more complicated case with $t_l \leq s < t_{l+1} \leq t_k \leq t < t_{k+1}$. In this case, equation 4.5 can be rewritten as

$$\begin{aligned}
I(t) = &\sum_{j=0}^{l-1} \Delta(t_j)[W(t_{j+1}) - W(t_j)] + \Delta(t_l)[W(t_{l+1}) - W(t_l)] \\
&+ \sum_{j=l+1}^{k-1} \Delta(t_j)[W(t_{j+1}) - W(t_j)] + \Delta(t_k)[W(t) - W(t_k)]
\end{aligned}$$

Now we take conditional expectation for each of these four terms in the right-hand side of the above equation in order to check the martingale property. As for the first

term, because $\sum_{j=0}^{l-1}\Delta(t_j)[W(t_{j+1})-W(t_j)]$ is $\mathcal{F}(s)$-measurable, we have

$$\mathbb{E}\Big[\sum_{j=0}^{l-1}\Delta(t_j)[W(t_{j+1})-W(t_j)]|\mathcal{F}(s)\Big]=\sum_{j=0}^{l-1}\Delta(t_j)[W(t_{j+1})-W(t_j)]. \tag{4.6}$$

As for the second term, because $\Delta(t_l)$ as well as $W(t_l)$ are $\mathcal{F}(s)$-measurable and the Brownian motion W is a martingale by theorem 1, we have

$$\begin{aligned}\mathbb{E}[\Delta(t_l)[W(t_{l+1})-W(t_l)]|\mathcal{F}(s)] &= \Delta(t_l)\mathbb{E}[W(t_{l+1})-W(t_l)|\mathcal{F}(s)]\\ &= \Delta(t_l)\big(\mathbb{E}[W(t_{l+1})|\mathcal{F}(s)]-\mathbb{E}[W(t_l)|\mathcal{F}(s)]\big)\\ &= \Delta(t_l)\big(W(s)-W(t_l)\big)\end{aligned} \tag{4.7}$$

As for the summands in the third term, because $\mathcal{F}(t_j)\subseteq\mathcal{F}(s)$, we can apply the tower property to get

$$\begin{aligned}\mathbb{E}\Big[\Delta(t_j)[W(t_{j+1})-W(t_j)]|\mathcal{F}(s)\Big] &= \mathbb{E}\Big[\mathbb{E}\big[\Delta(t_j)[W(t_{j+1})-W(t_j)]|\mathcal{F}(t_j)\big]|\mathcal{F}(s)\Big]\\ &= \mathbb{E}\Big[\Delta(t_j)\big(\mathbb{E}[W(t_{j+1})|\mathcal{F}(t_j)]-W(t_j)\big)|\mathcal{F}(s)\Big]\\ &= \mathbb{E}\Big[\Delta(t_j)\big(W(t_j)-W(t_j)\big)|\mathcal{F}(s)\Big]\\ &= 0.\end{aligned}$$

Hence, the third term as the summation of all the above summands should also vanish, that is,

$$\sum_{j=l+1}^{k-1}\Delta(t_j)[W(t_{j+1})-W(t_j)]=0. \tag{4.8}$$

Similarly, we can also apply the tower property to the fourth term,

$$\begin{aligned}\mathbb{E}\Big[\Delta(t_k)[W(t)-W(t_k)]|\mathcal{F}(s)\Big] &= \mathbb{E}\Big[\mathbb{E}\big[\Delta(t_k)[W(t)-W(t_k)]|\mathcal{F}(t_k)\big]|\mathcal{F}(s)\Big]\\ &= \mathbb{E}\Big[\Delta(t_k)\big(\mathbb{E}[W(t)|\mathcal{F}(t_k)]-W(t_k)\big)|\mathcal{F}(s)\Big]\\ &= \mathbb{E}\Big[\Delta(t_k)\big(W(t_k)-W(t_k)\big)|\mathcal{F}(s)\Big]\\ &= 0.\end{aligned} \tag{4.9}$$

By summing up equation 4.6, 4.7, 4.8 and 4.9, we get the conditional expectation $\mathbb{E}[I(t)|\mathcal{F}(s)]$,

$$\begin{aligned}\mathbb{E}[I(t)|\mathcal{F}(s)] &= \sum_{j=0}^{l-1}\Delta(t_j)[W(t_{j+1})-W(t_j)]+\Delta(t_l)\big(W(s)-W(t_l)\big)+0+0\\ &= I(s),\end{aligned}$$

which shows that the Itô's integral with simple integrands as defined by equation 4.5 is a martingale.

□

As defined in 4.5, we know $I(0)=0$, which is the initial information we know for sure, so we can get the expectation of $I(t)$,

$$\mathbb{E}[I(t)]=\mathbb{E}[I(t)|\mathcal{F}(0)]=I(0)=0.$$

Then, to obtain the variance of $I(t)$, we need to focus on $VarI(t) = \mathbb{E}I^2(t)$, which is evaluated in the following theorem.

Theorem 4.2.2. *The Itô's integral with simple integrands as defined by equation 4.5 has the variance*

$$\mathbb{E}I^2(t) = \mathbb{E}\int_0^t \Delta^2(s)ds. \tag{4.10}$$

Proof. For simplicity, we denote the Brownian increments as

$$D_j = W(t_{j+1}) - W(t_j), \forall j = 0, ..., k-1,$$

$$D_k = W(t) - W(t_k).$$

In this way, equation 4.5 can be rewritten as

$$I(t) = \sum_{j=0}^{k} \Delta(t_j)D_j,$$

so we have

$$I^2(t) = \sum_{j=0}^{k} \Delta^2(t_j)D_j^2 + 2\sum_{0\le i<j\le k} \Delta(t_i)\Delta(t_j)D_iD_j.$$

Our goal is to show that the expectation of the second term vanishes and the expectation of the first term is equal to the right-hand side of equation 4.10.
In the second term, given $i < j$, $\Delta(t_i)\Delta(t_j)D_i$ is $\mathcal{F}(t_j)$-measurable, D_j is independent of $\mathcal{F}(t_j)$, and the Brownian increment D_j has expectation 0, so we have

$$\mathbb{E}[\Delta(t_i)\Delta(t_j)D_iD_j] = \mathbb{E}[\Delta(t_i)\Delta(t_j)D_i]\cdot\mathbb{E}[D_j] = \mathbb{E}[\Delta(t_i)\Delta(t_j)D_i]\cdot 0 = 0,$$

which leads to

$$\mathbb{E}\Big[2\sum_{0\le i<j\le k} \Delta(t_i)\Delta(t_j)D_iD_j\Big] = 0.$$

Thus,

$$\begin{aligned}
\mathbb{E}I^2(t) &= \mathbb{E}\Big[\sum_{j=0}^{k} \Delta^2(t_j)D_j^2\Big] \\
&= \sum_{j=0}^{k} \mathbb{E}[\Delta^2(t_j)D_j^2] \\
&= \sum_{j=0}^{k} \mathbb{E}\Delta^2(t_j)\cdot\mathbb{E}D_j^2 \\
&= \sum_{j=0}^{k-1} \mathbb{E}\Delta^2(t_j)(t_{j+1}-t_j) + \mathbb{E}\Delta^2(t_k)(t-t_k) \\
&= \sum_{j=0}^{k-1} \mathbb{E}\int_{t_j}^{t_{j+1}} \Delta^2(s)ds + \mathbb{E}\int_{t_k}^{t} \Delta^2(s)ds \\
&= \mathbb{E}\int_0^t \Delta^2(s)ds,
\end{aligned}$$

where we have applied the facts that $\Delta(t_j)^2$ and D_j^2 are independent, $D_j = W(t_{j+1}) - W(t_j)$ has the distribution $N(0, t_{j+1} - t_j), \forall j$, and simple integrands $\Delta(t)$ are constant on the subintervals. □

Beside variance, another important quantity for Itô's integral is quadratic variation, which is evaluated in the following theorem.

Theorem 4.2.3. *The Itô's integral with simple integrands as defined by equation 4.5 has the quadratic variation*

$$[I, I](T) = \int_0^T \Delta^2(t)dt.$$

Proof. Taking a partition $\Pi = \{t_0 = 0, t_1, ..., t_n = T\}$ of $[0, T]$. Our goal is to first compute the quadratic variation accumulated on one subinterval $[t_j, t_{j+1}]$ on which $\Delta(t)$ is constant, and then sum over all the subintervals to get the total quadratic variation accumulated up to time T.
Take one of the subintervals $[t_j, t_{j+1}]$, on which $\Delta(t)$ is constant. We further take partition points inside of this subinterval.

$$t_j = s_0 < s_1 < ... < s_m = t_{j+1}, \tag{4.11}$$

so the quadratic variation accumulated on the subinterval $[t_j, t_{j+1}]$ can be written as

$$\begin{aligned}
\lim_{m\to\infty} \sum_{i=0}^{m-1} [I(s_{i+1}) - I(s_i)]^2 &= \lim_{m\to\infty} \sum_{i=0}^{m-1} [\Delta(t_j)(W(s_{i+1}) - W(s_i))]^2 \\
&= \lim_{m\to\infty} \Delta^2(t_j) \sum_{i=0}^{m-1} (W(s_{i+1}) - W(s_i))^2 \\
&= \Delta^2(t_j)(t_{j+1} - t_j) \\
&= \int_{t_j}^{t_{j+1}} \Delta^2(u)du,
\end{aligned}$$

where we have applied the fact that the quadratic variation of the Brownian motion $W(t)$ accumulated on the subinterval $[t_j, t_{j+1}]$ is by definition

$$\lim_{m\to\infty} \sum_{i=0}^{m-1} (W(s_{i+1}) - W(s_i))^2 = t_{j+1} - t_j.$$

Thus, by summing up the quadratic variations over all of the subintervals we get

$$[I, I](T) = \int_0^T \Delta^2(t)dt.$$

□

Now we can extend the above results to the Itô's integrals with general integrands, i.e., $\Delta(t)$ in equation 4.2.1 can be any stochastic process that satisfies the "square-integrability condition" [21]:

$$\mathbb{E} \int_0^T \Delta^2(t)dt < \infty. \tag{4.12}$$

Theorem 4.2.4. *The Itô's integral $I(t)$ with general integrands $\Delta(t)$ of $0 < t < T$ that satisfies 4.12 has the following properties:*

(i) $I(t)$ is a martingale;
(ii) The variance of $I(t)$ is $\mathbb{E}I^2(t) = \mathbb{E}\int_0^t \Delta^2(s)ds$;
(iii) The quadratic variation of $I(t)$ is $[I,I](t) = \int_0^t \Delta^2(s)ds$.

The above properties can be extended from theorem 4.2.1, 4.2.2 and 4.2.3 by implementing the standard machine from measure theory [22], which will not be elaborated here.

Definition 4.2.2 (Itô process). *Assume $W(t)$ is a Brownian motion with a filtration $\mathcal{F}(t)$ of $t \geq 0$. Then an Itô process is a random variable of the following form:*

$$X(t) = X(0) + \int_0^t \Delta(s)dW(s) + \int_0^t \Theta(s)ds,$$

where $X(0)$ is nonrandom and $\Delta(t)$ as well as $\Theta(t)$ are adapted stochastic processes.

Lemma 4.2.1. *Assume $X(t)$ of $t \geq 0$ is an Itô process, then its quadratic variation is*

$$[X,X](T) = \int_0^T \Delta^2(t)dt$$

Informal proof. The formal proof can be found in [21]. Here we provide an informal proof by applying the differential multiplication results $dW(t)dW(t) = t$, $dW(t)dt = dtdW(t) = 0$ and $dtdt = 0$. In this form, we have

$$\begin{aligned} dX(t)dX(t) &= \Delta^2(t)dW(t)dW(t) + 2\Delta(t)\Theta(t)dW(t)dt + \Theta^2(t)dtdt \\ &= \Delta^2(t)dt. \end{aligned}$$

□

Theorem 4.2.5 (Itô-Doeblin formula for an Itô process). *Assume $X(t)$ for $t \geq 0$ is an Itô process as defined in definition 4.2.2 and $f(t,x)$ is a function with defined and continuous partial derivatives $f_t(t,x)$, $f_x(t,x)$ and $f_{xx}(t,x)$. Then, $\forall T \geq 0$, we have*

$$\begin{aligned} &f(T, X(T)) \\ &= f(0, X(0)) + \int_0^T f_t(t, X(t))dt + \int_0^T f_x(t, X(t))dX(t) \\ &+ \frac{1}{2}\int_0^T f_{xx}(t, X(t))d[X,X](t) \\ &= f(0, X(0)) + \int_0^T f_t(t, X(t))dt + \int_0^T f_x(t, X(t))\Delta(t)dW(t) \\ &+ \int_0^T f_x(t, X(t))\Theta(t)dt + \frac{1}{2}\int_0^T f_{xx}(t, X(t))\Delta^2(t)dt. \end{aligned}$$

As the result is straightforward to understand, the rigorous proof is not provided here and can be found in [21] instead. For simplicity, the result can be rewritten in the differential form as follows:

$$\begin{aligned} df(t, X(t)) &= f_t(t, X(t))dt + f_x(t, X(t))dX(t) + \frac{1}{2}f_{xx}(t, X(t))dX(t)dX(t) \\ &= f_t(t, X(t))dt + f_x(t, X(t))\Delta(t)dW(t) + f_x(t, X(t))\Theta(t)dt + \frac{1}{2}f_{xx}(t, X(t))\Delta^2 dt \end{aligned}$$

Definition 4.2.3 (Multivariable Brownian motion). *Let $W_1(t), ..., W_d(t)$ be a series of one-dimensional Brownian motion which satisfies that $W_i(t)$ and $W_j(t)$ are independent if*

$i \neq j$. Then the process $W(t) = (W_1(t), ..., W_d(t))$ is a d-dimensional Brownian motion. There exists a filtration $\mathcal{F}(t)$ associated with $W(t)$ that satisfies
(i) for $0 \leq s < t$, $\mathcal{F}(s) \subseteq \mathcal{F}(t)$;
(ii) $\forall t \geq 0$, $W(t)$ is $\mathcal{F}(t)$-measurable;
(iii) for $0 \leq s < t$, $W(t) - W(s)$ is independent of $\mathcal{F}(s)$.

Now we can generalize the Itô-Doeblin formula in theorem 4.2.5 to the case with multidimensional Brownian motions.

Theorem 4.2.6 (Two-dimensional Itô-Doeblin formula)**.** *Assume $X(t)$ and $Y(t)$ for $t \geq 0$ are two Itô processes and $f(t, x, y)$ is a function with defined and continuous partial derivatives $f_t, f_x, f_y, f_{xx}, f_{xy}, f_{yx}$ and f_{yy}. Then, in the differential form the Itô-Doeblin formula is*

$$\begin{aligned} df(t, X(t), Y(t)) = {} & f_t(t, X(t), Y(t))dt + f_x(t, X(t), Y(t))dX(t) + f_y(t, X(t), Y(t))dY(t) \\ & + \frac{1}{2} f_{xx}(t, X(t), Y(t))dX(t)dX(t) + f_{xy}(t, X(t), Y(t))dX(t)dY(t) \\ & + \frac{1}{2} f_{yy}(t, X(t), Y(t))dY(t)dY(t). \end{aligned}$$

Note that here we can introduce an Itô product rule.

Corollary 4.2.1. *Assume $X(t)$ and $Y(t)$ are Itô processes. Then in the differential form we have*

$$d(X(t)Y(t)) = X(t)dY(t) + Y(t)dX(t) + dX(t)dY(t).$$

Again, the rigorous proof for the above two results are not provided here and can be found in [21].

Definition 4.2.4 (Stochastic differential equation)**.** *A stochastic differential equation is an equation of the form*

$$dX(t) = \beta(t, X(t))dt + \gamma(t, X(t))dW(t),$$

where $X(t)$is a stochastic process and the given functions $\beta(t, X(t))$ and $\gamma(t, X(t))$ are referred to the "drift" and "diffusion", respectively.

Normally, given certain conditions including the initial condition $X(0)$, we can solve the stochastic differential equations. Nevertheless it can be hard to solve them to obtain a closed-form solution. One of the most important stochastic models is given in the following example.

Example 4.2.1 (Geometric Brownian motion)**.** *The stochastic differential equation for geometric Brownian motion is*

$$dS(t) = \alpha S(t)dt + \sigma S(t)dW(t),$$

which has the solution

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

Chapter 5

Model in Uniform Calculation Method

In this chapter, the financial market model in URM is elaborated. As we mainly focus on the nominal term structures, the real term structures are omitted from this introduction. The main reference of this chapter is the CPB (Netherlands Bureau for Economic Policy Analysis) background document *A Financial Market Model for the Netherlands* by Nick Draper in 2014 [3].

This model estimates the interest rate, bond market and stock market in the Netherlands. The interest rate is modelled with two state variables, namely, X_{1t} and X_{2t}, which are incorporated in a 2-vector X_t. The state variables can be interpreted as the uncertainty and dynamics of the real interest rate and the instantaneous expected inflation. They follow a mean-reverting process around zero and are defined as

$$dX_t = -KX_t dt + \Sigma'_X dZ_t{}^{1},$$

where K is a 2×2 matrix, $\Sigma'_X = [I_{2\times 2} 0_{2\times 2}]$ [2], and Z is a 4-dimensional Brownian motion which drives the uncertainty in the market regarding the real interest rate, instantaneous expected inflation, unexpected inflation ad stock return[3]. We can express the two state variables separately as

$$dX_{1t} = -K_{11}X_{1t}dt - K_{12}X_{2t}dt + dZ_{1t},$$

$$dX_{2t} = -K_{21}X_{1t}dt - K_{22}X_{2t}dt + dZ_{2t}.$$

Then, the instantaneous nominal interest rate R_t is modelled as

$$R_t = R_0 + R'_1 X_t,$$

where the parameter R_0 is the value of the initial nominal interest rate, and R_1 is a 2-vector.

A risk premium is the excess return obtained by the investor for taking risks. The risk premium Λ_t is modelled as

$$\Lambda_t = \Lambda_0 + \Lambda_1 X_t, \tag{5.1}$$

[1] In this chapter, the prime symbol denotes the matrix transposition, for example, $A' = A^T$.

[2] $\Sigma'_X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$.

[3] Even though the last two entries Z_{3t} and Z_{4t} are cancelled out due to Σ'_X while modelling the state variables, they are taken into consideration later while modelling the stock prices.

where Λ_0 is a 2-vector and Λ_1 is a 2×2 matrix. we can express the two terms in Λ_t separately as

$$\Lambda_{1t} = \Lambda_{0(1)} + \Lambda_{1(11)} X_{1t} + \Lambda_{1(12)} X_{2t},$$
$$\Lambda_{2t} = \Lambda_{0(2)} + \Lambda_{1(21)} X_{1t} + \Lambda_{1(22)} X_{2t}.$$

The present value of a future cash flow, such as the prices of stocks and zero coupon bonds, can be computed by multiplying the future value by a discount factor and then taking the expectation. The nominal stochastic discount factor[4] ϕ_t can be modeled as

$$\frac{d\phi_t}{\phi_t} = -R_t dt - \Lambda_t' dZ_t, \tag{5.2}$$

where Λ_t is extended to the 4-vector $[\Lambda_{1t}, \Lambda_{2t}, 0, 0]'$ to match the dimension of Z_t. The stock index S is modelled by a geometric Brownian motion,

$$\frac{dS_t}{S_t} = (R_t + \eta_S)dt + \sigma_S' dZ_t, \tag{5.3}$$

where the parameter $\sigma_S' \in \mathbb{R}^4$, $S_0 = 1$ and η_S denotes the equity risk premium, namely, the risk premium of investments in the stock market.

In this thesis, the following discrete approximation of the stochastic differential equation 5.3 is implemented to compute the stock returns in the URM model,

$$\frac{S(t_{i+1})}{S(t_i)} = \exp\{(R_t + \eta_S - \frac{1}{2}\sigma_S'\sigma_S)\Delta t + \sigma_S' V\}, \tag{5.4}$$

where $V' = [V_1, V_2, V_3, V_4]$ is a 4-vector consisting of 4 independent normal random variables $V_i \sim N(0, \Delta t), \forall i \in \{1, 2, 3, 4\}$.

Note that there are some parameter restrictions imposed on the model. We have the fundamental pricing equation for the stock index

$$\mathbb{E} d\phi S = 0, \tag{5.5}$$

which formulates that the expected value of the discounted stock price does not change over time.
According to corollary 4.2.1,

$$\begin{aligned}\frac{d\phi S}{\phi S} &= \frac{d\phi}{\phi} + \frac{dS}{S} + \frac{d\phi}{\phi} \cdot \frac{dS}{S} \\ &= (-R_t dt - \Lambda_t' dZ_t) + ((R_t + \eta_S)dt + \sigma_S' dZ_t) \\ &\quad + (-R_t dt - \Lambda_t' dZ_t) \cdot ((R_t + \eta_S)dt + \sigma_S' dZ_t).\end{aligned}$$

According to the differential multiplication from 4.4, we further have

$$\frac{d\phi S}{\phi S} = (\eta_S - \Lambda_t'\sigma_S')dt - (\Lambda_t' - \sigma_S')dZ_t.$$

[4]The "nominal" here emphasizes that the discount factor is modelled with the instantaneous nominal interest rate R_t.

Taking the expectation and using equation 5.5, we get the restriction

$$\eta_S = \Lambda_t' \sigma_S. \tag{5.6}$$

Substituting equation 5.1 into 5.6, we obtain

$$\eta_S = \Lambda_0' \sigma_S + X_t' \Lambda_1' \sigma_S,$$

which implies $\Lambda_0' \sigma_S = \eta_S$ and $\sigma_S' \Lambda_1 = 0$.

We have the fundamental pricing equation for a nominal zero coupon bond

$$\mathbb{E} d\phi P = 0,$$

which means that the expected value of the discounted nominal bond price does not change over time. According to corollary 4.2.1,

$$\mathbb{E} d\phi P = \mathbb{E}[P d\phi + \phi dP + d\phi \cdot dP] = 0. \tag{5.7}$$

Assume the nominal zero coupon bond price is a function of time and the state of economy, i.e., $P = P(X, t)$, then by Itô-Doeblin formula as in theorem 4.2.5 we have [5]

$$\begin{aligned} dP &= P_X' dX + P_t dt + \frac{1}{2} dX' P_{XX'} dX \\ &= P_X'(-KX_t dt + \Sigma_X' dZ_t) + P_t dt + \frac{1}{2} dZ_t' \Sigma_X P_{XX'} \Sigma_X' dZ_t \\ &= [P_X'(-KX_t) + P_t + \frac{1}{2} \Sigma_X P_{XX'} \Sigma_X'] dt + P_X' \Sigma_X' dZ_t. \end{aligned} \tag{5.8}$$

Substituting equation 5.2 and 5.8 into 5.7, we know that the dt term should vanish to keep the expectation zero, that is,

$$P_X'(-KX_t) + P_t + \frac{1}{2} tr(\Sigma_X P_{XX'} \Sigma_X') - PR_t - P_X' \Sigma_X' \Lambda_t = 0. \tag{5.9}$$

Below we shall show that a solution of this partial differential equation is of the form

$$P(X_t, t, t+\tau) = \exp(A(\tau) + B(\tau)' X_t), \tag{5.10}$$

where τ is the duration to maturity and is defined as $\tau = T - t$ with T the maturity time.

Now we try to obtain the explicit expressions of function $A(\tau)$ and $B(\tau)$. From equation 5.10 we can get

$$\frac{1}{P} P_X = B,$$

$$\frac{1}{P} P_{XX'} = BB',$$

[5] Note that in this chapter the prime symbol denotes the matrix transposition, for example, $A' = A^T$. Here, P_X, P_{XX} and P_t denote the first derivative of P regarding X, the second derivative pf P regarding X and the first derivative of P regarding t, respectively. Consistent with earlier, X_t and Z_t are not derivatives, and they simply denote that the state variables and the associated Brownian motion are time-dependent. Besides, Σ_X is not a derivative either, and it is defined as the matrix $[I_{2\times 2} 0_{2\times 2}]'$ earlier.

$$\frac{1}{P}P_t = -\frac{1}{P}P_\tau = -\dot{A} - \dot{B}'X_t,$$

where $\dot{A}$ denotes the derivative $\frac{dA}{d\tau}$ and $\dot{B}$ denotes the derivative $\frac{dB}{d\tau}$.
Note that the third equation is from the fact that $d\tau = -dt$. Substituting the above derivatives into equation 5.9 we can get

$$B'(-KX_t) + (-\dot{A} - \dot{B}'X_t) + \frac{1}{2}tr(\Sigma_X BB'\Sigma_X') - R_0 - R_1'X_t - B'\Sigma_X'(\Lambda_0 + \Lambda_1 X_t) = 0,$$

where $tr(\Sigma_X BB'\Sigma_X') = tr(B'\Sigma_X'\Sigma_X B) = B'\Sigma_X'\Sigma_X B$.

Because both the stochastic term and the non-stochastic term have to be zero, we get

$$\dot{A}(\tau) = -R_0 - (\Lambda_0'\Sigma_X)B(\tau) + \frac{1}{2}B'(\tau)\Sigma_X'\Sigma_X B(\tau),$$

$$\dot{B}(\tau) = -R_1 - (K' + \Lambda_1'\Sigma_X)B(\tau).$$

By definition of the zero coupon bond, we know the bond price should be 1 when the duration to maturity $\tau = 0$, that is, $P(X_t, t, t) = 1$, which leads to $A(0) = B(0) = 0$. The solution to the above differential equations can be obtained,

$$B(\tau) = (K' + \Lambda_1'\Sigma_X)^{-1}[\exp(-(K' + \Lambda_1'\Sigma_X)\tau) - I_{2\times 2}]R_1,$$

$$A(\tau) = \int_0^\tau \dot{A}(s)ds.$$

Thus, we solve the function A and B in the closed form for the nominal zero coupon bond index as described in equation 5.10.

Assume the portfolio is permanently rebalanced so that the duration to maturity τ of the bonds that are invested is kept constant. In this case, we denote the nominal bond price with constant maturity τ as P^τ. substituting $P_t^\tau = -P_\tau^\tau$ into equation 5.8 we obtain

$$dP^\tau = [P_X^{\tau'}(-KX_t) + \frac{1}{2}\Sigma_X P_{XX'}^\tau \Sigma_X']dt + P_X^{\tau'}\Sigma_X' dZ_t,$$

which leads to

$$\frac{dP^\tau}{P^\tau} = [-B'KX_t + \frac{1}{2}B'\Sigma_X'\Sigma_X B]dt + B'\Sigma_X' dZ_t. \tag{5.11}$$

According to the fundamental pricing equation 5.7, we have the restriction

$$-B'KX_t + \frac{1}{2}B'\Sigma_X'\Sigma_X B - R_t - B'\Sigma_X'\Lambda_t = 0. \tag{5.12}$$

Thus, substituting equation 5.12 into 5.11, we arrive at the "funds price dynamics equation",

$$\frac{dP^\tau}{P^\tau} = [R_t + B'\Sigma_X'\Lambda_t]dt + B'\Sigma_X' dZ_t. \tag{5.13}$$

In this thesis, the long-term interest rate is approximated by the long-term treasury rate, which means that we can compute the annual nominal interest rate from the nominal bond prices. Assume only the bonds with duration to maturity τ are invested, then the corresponding annual interest rate can be computed as $(\frac{1}{P})^{\frac{1}{\tau}} - 1$.

The following discrete approximation of equation 5.13 is implemented to compute the bond returns in the URM model,

$$\frac{P^{\tau}(t_{i+1})}{P^{\tau}(t_i)} = \exp\{(R_t + B'\Sigma_X'\Lambda_t - \frac{1}{2}B'\Sigma_X'\Sigma_X B)\Delta t + B'\Sigma_X' V\}, \tag{5.14}$$

where V is the 4-vector as designated in equation 5.4.

Thereby we have introduced all the quantities that are relevant for the pension estimation for this thesis, namely, instantaneous interest rate, stock market and bond market.

The parameters are then calibrated based on the data from [25]. Two possible sets of parameters can be concluded in Table 5.1.

TABLE 5.1: Parameters estimated for the Netherlands in the financial market model in URM [3]

Parameter	Estimate	(SD)	Estimate	(SD)
Nominal interest rate R_t				
R_0	2.40%	(6.06%)	2.53%	(4.86%)
$R_{1(1)}$	-1.48%	(0.22%)	1.29%	(0.32%)
$R_{1(2)}$	0.53%	(0.56%)	0.91%	(0.41%)
State variables X_t				
κ_{11}	0.08	(0.11)	0.35	(0.19)
κ_{22}	0.35	(0.18)	0.08	(0.10)
κ_{21}	-0.19	(0.08)	-0.20	(0.17)
Stock return process $\frac{dS_t}{S_t}$				
η_S	4.52%	(3.73%)	4.54%	(3.73%)
$\sigma_{S(1)}$	-0.53%	(1.44%)	-0.32%	(1.59%)
$\sigma_{S(2)}$	-0.76%	(1.54%)	0.88%	(1.52%)
$\sigma_{S(3)}$	-2.11%	(1.51%)	-2.09%	(1.52%)
$\sigma_{S(4)}$	16.59%	(0.96%)	16.60%	(0.95%)
Prices of risk Λ_t				
$\Lambda_{0(1)}$	0.403	(0.333)	-0.200	(0.313)
$\Lambda_{0(2)}$	0.039	(0.270)	-0.347	(0.266)
$\Lambda_{1(1,1)}$	0.149	(0.156)	0.135	(0.218)
$\Lambda_{1(1,2)}$	-0.381	(0.039)	-0.080	(0.150)
$\Lambda_{1(2,1)}$	0.089	(0.075)	0.401	(0.183)
$\Lambda_{1(2,2)}$	-0.083	(0.129)	-0.068	(0.121)

Chapter 6

Foundation for Deep Learning

In this chapter, the foundation for deep learning is presented. Firstly, the basic concepts in deep learning theory and artificial neural networks are introduced, followed by the elaboration of the backpropagation algorithm, one of the most important underlying mechanisms of deep learning. Then, major challenges and techniques in deep learning are discussed. As the focus of this thesis project, the convolutional neural network and recurrent neural network are presented in detail. Last but not least, an innovative and successful neural network architecture named encoder-decoder architecture is briefly addressed. The main references of this chapter include *Neural Networks and Deep Learning* by Michael A. Nielsen [16] and *Deep Learning* by Ian Goodfellow, et al. [5].

6.1 Introduction to Deep Learning

Artificial intelligence has been a thriving topic nowadays, and lying in the heart is the active research field of machine learning, which is the scientific study of building a model by learning model parameters from data of previous observations in order to make predictions or classifications for future or further data. There are mainly two categories of machine learning tasks, namely, supervised and unsupervised learning. Supervised machine learning aims to learn a function that maps the given input-output data pairs, i.e., labels. Typical supervised learning includes regression and classification. Unsupervised learning refers to the learning tasks without labels, such as denoising and imputation of missing values. There are many machine learning techniques that are heavily applied in industries, including logistic regression, nearest neighbor classification, support vector machines, decision trees and random forest, etc. Specifically, a rapid increase in implementing deep learning theory has been seen since the last decade.

The idea of deep learning is to develop non-task-specific algorithms to enable computers to automate the process of learning tasks without requiring rule-based hard coding based on the expertise in the relevant field or too much human intervention to manipulate the data. Inspired by the biological nervous system, artificial neural networks are constructed to carry out the deep learning theory. The basic architecture of deep neural network is introduced below.

Definition 6.1.1 (Artificial neuron)**.** *Artificial neurons are the basic elements of artificial neural networks. A neural network consists of several layers of artificial neurons, each of which maps from one or several inputs to an output.*

Definition 6.1.2 (Perceptron)**.** *A perceptron is a type of neuron mapping from one or several binary inputs to a single binary output. The output of a perceptron is determined by the*

weighted sum of inputs $\sum_j w_j x_j + b$ with bias b and weights w_i corresponding to inputs x_i in the following way:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b \leq 0, \\ 1 & \text{if } \sum_j w_j x_j + b > 0. \end{cases}$$

Example 6.1.1. *To illustrate the usage of perceptrons, we construct a decision-making model consisting of one perceptron [16]. Assume we decide if we go to a concert by weighing up the following three factors:*

1. Is the band holding the concert of interest?
2. Is the ticket of the concert within budget?
3. Is the venue of the concert easily reachable?

We can use three binary variables x_1, x_2 and x_3 to represent the above three factor, respectively. If the response to a factor is "yes", then the corresponding variable is set to be 1, otherwise 0. Based on our preference regarding the significance of each of the three factors, we can set the values of the corresponding weights w_1, w_2 and w_3. Based on our general motivation level of going to a concert, we can set the value of the associated bias b.

Suppose we feel the first factor is the most important one, while the second is the least we care about, then we may set the three weights to be $w_1 = 4$, $w_2 = 1$ and $w_3 = 2$. Besides, if we are enthusiastic about going to concerts in general, then we may set the bias to be a relatively small number such as $b = -3$. In the case that the band holding the concert is of interest, the ticket is within in budget, whereas the venue is not easily reachable, the output can be computed to be equal to 1, as $4 \times 1 + 1 \times 1 + 2 \times 0 - 3 = 2 > 0$.

This perceptron with three inputs x_1, x_2 and x_3 can be illustrated in figure 6.1.

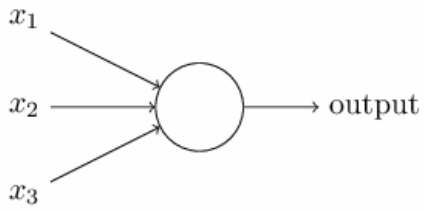


FIGURE 6.1: A perceptron with three inputs x_1, x_2 and x_3 [16].

If our network consists of perceptrons, then inputs and outputs have to be binary, and changes in weights and bias may cause a complete flip in the output, which imposes limitations to the network and difficulties in training robustly. To solve this problem, other types of neurons are defined.

Definition 6.1.3 (Sigmoid neuron)**.** *A sigmoid neuron is a type of neuron mapping from one or several input values in the range of $[0, 1]$ to a single output value in the range of $(0, 1)$. The output of a sigmoid neuron is determined by the sigmoid function σ acting on the weighted sum of inputs $z = \sum_j w_j x_j + b$ in the following way:*

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} = \frac{1}{1 + \exp(-\sum_j w_j x_j - b)} \quad ,$$

where the sigmoid function has the following shape:

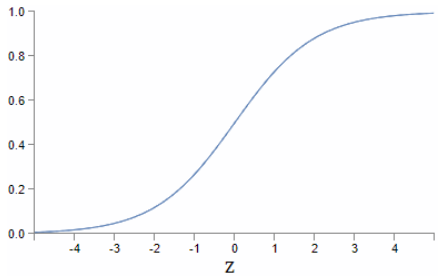


FIGURE 6.2: The shape of the sigmoid function $\sigma(z)$ [16].

Definition 6.1.4 (Activation function). *In an artificial neuron, the activation function is a function acting on the weighted sum of inputs $z = \sum_j w_j x_j + b$ of this neuron so as to compute the output of the neuron.*

The sigmoid function is one of the most used activation functions. By definition 6.1.2 and 6.1.4, the activation function $a(z)$ of the aforementioned perceptrons is actually a step function

$$a(z) = \begin{cases} 0 & \text{if} \quad z \leq 0, \\ 1 & \text{if} \quad z > 0, \end{cases}$$

which has the following shape:

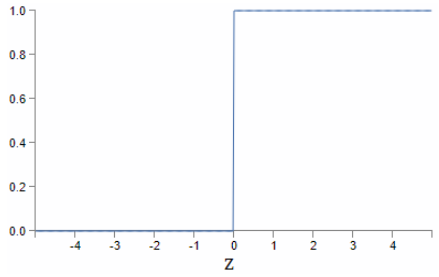


FIGURE 6.3: The shape of the step function as the activation function of perceptrons [16].

Besides perceptrons and sigmoid neurons, there are also other basic and popular artificial neurons with alternative activation functions.

Definition 6.1.5 (Tanh neuron). *Tanh neuron is a type of neuron with tanh function as its activation function, which is defined by*

$$\tanh(z) \equiv \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

where $z = \sum_j w_j x_j + b$ is the weighted sum of inputs.

It follows that

$$\sigma(z) = \frac{1 + \tanh(z/2)}{2},$$

which infers that tanh function can be obtained by rescaling sigmoid function. The shape of tanh function is also consistent with this observation:

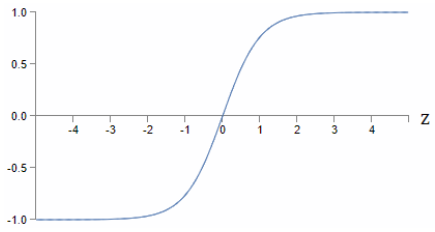


FIGURE 6.4: The shape of tanh function [16].

Definition 6.1.6 (Rectified linear neuron/unit (ReLU)). *Rectified linear neuron/unit (ReLU) is a type of neuron with rectifying function as its activation function, which is defined by*

$$\text{output} = \max(0, z) = \max(0, \sum_j w_j x_j + b).$$

The shape of the rectifying function is

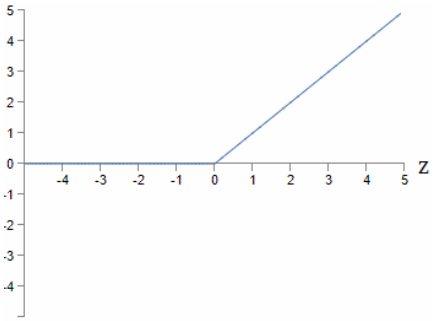


FIGURE 6.5: The shape of rectifying function [16].

Neurons are organized in layers in neural networks. In general, there are three categories of layers in one neural network, namely, one input layer, one or multiple hidden layers and one output layer. The input layer is the first layer that consists of input neurons. The output layer is the last layer that consists of output neurons. The hidden layers simply refer to all the layers between the input layer and output layer. Softmax layer is a type of commonly-used output layer.

Definition 6.1.7 (Softmax layer). *Softmax layer is a type of output layer with softmax function as its activation function, which is defined as*

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$

with a_j^L and z_j^L denoting the activation and the weighted sum of inputs of the j^{th} neuron in the output layer L, respectively.

An outstanding property of softmax layer is that the outputs are all lying in the range of $(0, 1]$ and summed up to 1, which resembles a probability distribution.

After building up the architecture of the neural network, we need to find a method to compare and evaluate the generated outputs with the given labels, which leads to the introduction of cost functions.

Definition 6.1.8 (Cost function). *A cost function $C(w,b)$ is defined as a function of all the weights w and biases b in a neural network, and it reflects the deviation of the generated outputs a from the given labels $y(x)$. There are two requirements for the cost function:*
(i) $C(w,b)$ decreases as outputs get closer to labels on average over all the training points;
(ii) the cost function C of the neural network is an average of the costs C_x over each of the training data points x, i.e., $C = \frac{1}{n}\sum_x C_x$, where n is the total number of training points.

Just like the activation functions, there are also multiple frequently-used cost functions, each of which has its own advantages under certain conditions.

Definition 6.1.9 (Quadratic cost function). *The quadratic cost function is defined as*

$$C(w,b) = \frac{1}{2n}\sum_x \|y(x) - a\|^2,$$

where n is the number of training inputs x, $y(x)$ is the vector of the true label corresponding to the input x, and a is the vector of the output generated by the neural network. For the quadratic cost function, $C_x = \frac{1}{2}\|y(x) - a\|^2$.

The quadratic cost function is also known as the mean squared error (MSE). Such cost functions are used by the machine as an indicator that steers the training and guides the tuning of the parameters in the network. The general idea is to change the weights and biases in such a way that the value of the cost function can be decreased, which leads to gradient descent, the central algorithm in deep learning theory.

Definition 6.1.10 (Gradient). *The gradient of the cost function C is defined as the vector*

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, ..., \frac{\partial C}{\partial v_n}\right)^T,$$

where v_i for $i = 1, ..., n$ denote all the parameters including weights and biases in the neural network.

Definition 6.1.11 (Gradient descent). *Gradient descent is an algorithm that requires the computer to repeatedly compute the gradient of the cost function and then change the parameters in the neural network in the opposite direction of ∇C according to the following update rule:*

$$w_k \to w_k' = w_k - \eta\frac{\partial C}{\partial w_k},$$

$$b_l \to b_l' = b_l - \eta\frac{\partial C}{\partial b_l},$$

where w_k denote all the weights, b_l denote all the biases, C is the cost function and $\eta > 0$ is a positive constant known as the learning rate.

Theorem 6.1.1. *The gradient descent algorithm as described in definition 6.1.11 can decrease the value of the cost function.*

Proof. Assume in the neural network the cost function is C and all the parameters including weights and biases are incorporated in the vector v. By calculus we have

$$\Delta C \approx \nabla C \cdot \Delta v,$$

where Δf denotes a small change in the function f.

Suppose we change the parameters in the opposite direction from ∇C, which can be denoted as

$$\Delta v = -\eta \nabla C,$$

where $\eta > 0$ is a constant.
Then we have

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2 \leq 0,$$

which indicates that the value of the cost function decreases. □

Gradient descent requires the computer to compute the gradient of cost ∇C_x for each of the training data x and then take the average $\nabla C = \frac{1}{n} \sum_x \nabla C_x$. This can be quite time-consuming and can cause the parameters not to be updated frequently enough. In practice, a modified algorithm called stochastic gradient decent is implemented instead to speed up learning.

Definition 6.1.12 (Stochastic gradient descent)**.** *Stochastic gradient descent is to randomly split the training set into many mini-batches, compute the average gradient of cost within each batch instead of the entire training set, and then update the parameters after going through each batch.*

In the next section the algorithm of computing the gradients of the cost function is discussed.

6.2 Backpropagation Algorithm

The workhorse of deep learning is the backpropagation algorithm, which is a breakthrough regarding computing the gradients of cost functions and making the realization of gradient descent feasible.

To neatly introduce the backpropagation algorithm, we first set the notations in the neural network. We use w^l_{jk} to denote the weight of the input from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. We use b^l_j, a^l_j and z^l_j to denote the bias, the activation and the weighted sum of the j^{th} neuron in the l^{th} layer, respectively.

Then, taking the sigmoid function σ as the activation function, we have

$$a^l_j = \sigma(z^l_j) = \sigma\Big(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\Big), \tag{6.1}$$

where the sum is over all the neurons in the $(l-1)^{th}$ layer that connect to this j^{th} neuron in the l^{th} layer.

For simplicity, we rewrite all the quantites in the matrix form. We use w^l to denote the weight matrix that incorporates all the weights of the inputs from the $(l-1)^{th}$ layer to the l^{th} layer. The $(j,k)^{th}$ entry of the weight matrix w^l is defined to be w^l_{jk}. Moreover, we use b^l, a^l and z^l to denote the weight vector, the bias vector and the vector of weighted sum in the l^{th} layer, in which b^l_j, a^l_j and z^l_j are the j^{th} entry, respectively. Besides, the vectorization of the function can be denoted as $\sigma(v)_j = \sigma(v_j)$, that is, the function σ on the vector v is applied to every element v_j of the vector.

Thus, we can rewrite equation 6.1 in the matrix form as

$$a^l = \sigma(z^l) = \sigma(w^l a^{l-1} + b^l).$$

The vectorization of the expression is more compact and straightforward. Moreover, in practice, matrix operations can be implemented fast in the matrix libraries in Python. A less common but important matrix operation in backpropagation is the Hadamard product.

Definition 6.2.1 (Hadamard product)**.** *The Hadamard product is the elementwise multiplication between vectors of the same dimension, which can be denoted as*

$$(v \odot u)_j = v_j u_j,$$

where v and u are two vectors of the same dimension.

The backpropagation algorithm enables the computation of gradients of the cost function C_x for each individual training input x, so below in this section we denote the cost function associated with each training input C_x as C for simplicity. Backpropagation consists of four fundamental equations, in which a central quantity, the error δ_j^l of the j^{th} neuron in the l^{th} layer, is defined as

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l},$$

where C is the cost function and z is the weighted sum of inputs.

Theorem 6.2.1 (BP1)**.** *The error δ^L in the output layer L can be computed by*

$$\delta^L = \nabla_a C \odot \sigma'(z^L).$$

Proof.

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L).$$

□

Theorem 6.2.2 (BP2)**.** *The error δ^l in the l^{th} layer can be computed through the value of the error δ^{l+1} in the next layer as*

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l).$$

Proof.

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l}\delta_k^{l+1}.$$

We have

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1}\sigma(z_j^l) + b_k^{l+1},$$

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1}\sigma'(z_j^l).$$

Thus, we finish proving

$$\delta^l = \sum_k w_{kj}^{l+1}\delta_k^{l+1}\sigma'(z_j^l).$$

□

Theorem 6.2.3 (BP3). *The partial derivative of the cost with respect to the biases in the network can be computed by*

$$\frac{\partial C}{\partial b} = \delta.$$

Proof.

$$\frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial C}{\partial z_k^l}\frac{\partial z_k^l}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l.$$

□

Theorem 6.2.4 (BP4). *The partial derivative of the cost with respect to the weights in the network can be computed by*

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l.$$

Proof.

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l.$$

□

The main idea of backpropagation is to obtain the gradients through computing the error δ backward through the layers from the output layer. Based on the four fundamental equations, we can write out the backpropagation algorithm 1.

Algorithm 1 Backpropagation algorithm [16]

1: **Input** x: set the corresponding activation a^1 for the input layer
2: **Feedforward**: for each $l = 2, 3, ..., L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3: **Output error** δ^L: compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4: **Backpropagate the error:** For each $l = L-1, L-2, ..., 2$ compute $\delta^l = ((w^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l)$.
5: **Output**: the gradient of the cost function is given by $\frac{\partial C}{\partial w_{jk^l}} = a_k^{l-1}\delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

Algorithm 2 Stochastic gradient descent (SGD) [16]

1: **Input a mini-batch of** m **training examples**
2: **For each training example x**:
3: **Input** x: set the corresponding activation a^1 for the input layer
4: **Feedforward**: for each $l = 2, 3, ..., L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.
5: **Output error** $\delta^{x,L}$: compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
6: **Backpropagate the error**: for each $l = L-1, L-2, ..., 2$ compute $\delta^{x,l} = ((w^{l+1})^T\delta^{x,l+1}) \odot \sigma'(z^{x,l})$.
7: **Gradient descent**: for each $l = L-1, L-2, ..., 2$ update the weights according to the rule $w^l \to w^l - \frac{\eta}{m}\sum_x \delta^{x,l}(a^{x,l-1})^T$, and the biases according to the rule $b^l \to b^l - \frac{\eta}{m}\sum_x \delta^{x,l}$.

The algorithm 1 computes gradients of the cost function $C = C_x$ for each individual training input x, which is usually combined with the algorithm 2 of stochastic gradient descent (SGD) in practice.

Definition 6.2.2 (Epoch)**.** *An epoch is defined as one round of passing through all the training data.*

Thus, when we train the neural network, there should be an outer loop generating batches and another outer loop going through a certain number of epochs.

As an alternative to the most basic and common method of SGD, another popular method for stochastic optimization is adaptive moment estimation (Adam) [11], which is briefly introduced here. As indicated by the name, this optimization method computes adaptive learning rates for each parameter by taking advantage of the first and second moments of gradients. Despite the surging popularity of Adam, it remains controversial if Adam keeps outperforming SGD [10], and the choice regarding the optimization methods can depend on the specific settings and the empirical results.

Algorithm 3 Adaptive moment estimation (Adam) [11]
Note that g_t^2 denotes $g_t \odot g_t$, β_1^t and β_2^t denote β_1 and β_2 to the power t, and all the operations on vectors are element-wise.

Set hyperparameters:
    α: Stepsize
    $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
    $C(\theta)$: Cost function
Initialize:
    θ_0: Initial parameter vector
    $m_0 \leftarrow 0$: Initial 1^{st} moment vector
    $v_0 \leftarrow 0$: Initial 2^{nd} moment vector
    $t \leftarrow 0$: Initial timestep
While θ_t not converged **do**:
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta C_t(\theta_{t-1})$ (Get gradients at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ ((Update biased second raw moment estimate)
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
End while
Return θ_t

6.3 Challenges and Techniques

In this section, we briefly discuss several common problems encountered while building a neural network as well as multiple powerful techniques that have been heavily implemented to help solve the problems.

6.3.1 Learning Slowdown Problem

Learning slowdown of a neural network refers to the problem that the parameters are updated very slowly and the error of the outcomes are not decreased rapidly enough during training. There can be multiple reasons that cause the learning slowdown problem, whereas in this section we mainly focus on the selection of activation and cost functions.

Assume a neural network consists of layers of sigmoid neurons as defined in 6.1.3, and the quadratic cost function as defined in definition 6.1.9 is applied. Then the gradients of the cost function can be explicitly written out as

$$\frac{\partial C}{\partial w} = (a - y)\sigma'(z)x,$$

$$\frac{\partial C}{\partial b} = (a - y)\sigma'(z).$$

We can see that the term $\sigma'(z)$ appears in both of the equations. By definition of the sigmoid function, when the weighted input z is close to 0 or 1, $\sigma'(z)$ is close to 0, which causes $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ to vanish. In this case, the learning slowdown occurs. To fix this problem, alternative activation and cost functions can be considered. Here as an example we introduce the commonly-used cross-entropy cost function.

Definition 6.3.1 (Cross-entropy cost function)**.** *The cross-entropy cost function is defined as*

$$C = -\frac{1}{n}\sum_x \sum_j [y_j \ln a_j^L + (1 - y_j)\ln(1 - a_j^L)].$$

Assume a neural network consists of layers of sigmoid neurons and the cross-entropy cost function is applied. We have

$$\begin{aligned}\frac{\partial C}{\partial w_j} &= -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\frac{\partial \sigma}{\partial w_j} \\ &= \frac{1}{n}\sum_x \frac{\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))}(\sigma(z) - y) \\ &= \frac{1}{n}\sum_x x_j(\sigma(z) - y),\end{aligned}$$

where we have used the fact that $\sigma'(z) = \sigma(z)(1 - \sigma(z))$. Similarly,

$$\frac{\partial C}{\partial b_j} = \frac{1}{n}\sum_x (\sigma(z) - y).$$

We can see that $\sigma'(z)$ is not involved in the expressions of gradients, so it is harder for the gradients in this network to vanish than in the previous one with the quadratic cost function, which helps mitigate the learning slowdown problem.

6.3.2 Overfitting Problem and Regularization

Overfitting is one of the major problems in training modern neural networks. It refers to the phenomenon that the neural network fits to the peculiarities of the training data too much to be able to make a generalization to the test data or any

further unseen data. One of the solution to overfitting is to apply the techniques of regularization. In this section, four categories of regularization methods are briefly introduced, namely, early stopping, L1 and L2 regularization, dropout and artificial expansion of training data.

In scientific researches, test data set should not be used during the training process, and should only be used after the model is finalized in order to evaluate the final performance of the model. However, we can separate a subset from the training set as the validation data set in order to monitor the performance of the network and help tune hyperparameters during training. Empirically, when the network is trained for too many epochs, the training error will keep decreasing, the network tends to get overfitting on the training data, and the test error will keep increasing due to the incapability of generalization. To resolve this problem, the technique of early stopping is introduced.

Definition 6.3.2 (Early stopping)**.** *Early stopping refers to the strategy that the error on the validation data is monitored and the training process is stopped before the validation error stops decreasing.*

In practice, while applying early stopping, the exact moment to stop training depends on the specific learning task and the researcher's personal preference of adopting aggressive strategies or not.
Besides early stopping, another essential technique is regularization, which is basically adding a regularization term on top of the original cost function to punish large weights.

Definition 6.3.3 (L2 regularization)**.** *L2 regularization is to change the original cost function C_0 to*

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2,$$

where $\lambda > 0$ is a constant known as the regularization parameter, n is the number of the training data points and w incorporates all the weight terms.

After regularizing the cost function, we have the new gradients

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n}w,$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}.$$

According to the gradient descent algorithm, the parameters should be updated following

$$w \to (1 - \frac{\eta\lambda}{n})w - \eta\frac{\partial C_0}{\partial w},$$

$$b \to b - \eta\frac{\partial C_0}{\partial b}.$$

Assume the training data set is split into multiple mini-batches of size m. According to stochastic gradient descent, the parameters should be updated following

$$w \to (1 - \frac{\eta\lambda}{n})w - \frac{\eta}{m}\sum_x \frac{\partial C_{0,x}}{\partial w},$$

$$b \to b - \frac{\eta}{m} \sum_x \frac{\partial C_0}{\partial b},$$

where $C_{0,x}$ is the original cost function for training data point x and the sums are over every batch. Comparing with the original unregularized updating rule, there is an extra term $1 - \frac{\eta\lambda}{n}$ multiplying to the weights, which is known as the weight decay factor.

Similarly, another technique of regularization can be introduced.

Definition 6.3.4 (L1 regularization)**.** *L1 regularization is to change the original cost function C_0 to*

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|,$$

where $\lambda > 0$ is a constant known as the regularization parameter, n is the number of the training data points and w incorporates all the weight terms.

In this case, we have the new gradients

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} sgn(w),$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b},$$

where $sgn(w)$ is the sign function of w.
Then the parameter updating rule is

$$w \to w - \frac{\eta\lambda}{n} sgn(w) - \eta \frac{\partial C_0}{\partial w},$$

$$b \to b - \eta \frac{\partial C_0}{\partial b}.$$

It is hard to prove that L1 and L2 regularization can help reduce overfitting, even though empirically this is the case in general. A persuading and concrete explanation is that by punishing the large weights the complexity is also lowered in the network, which helps with generalization.

Besides modifying the cost function, there are also techniques focusing on the architecture of the network.

Definition 6.3.5 (Dropout)**.** *While implementing dropout with the hyperparameter of probability p, each of the neurons in the hidden layers of the network can be temporarily deleted with probability p at the beginning of the training over a mini-batch, then the remaining weights and biases can be updated during the training. After going through a mini-batch, the deleted neurons are restored, all of the neurons in the hidden layers can be temporarily deleted with probability p again, and the remaining parameters can be updated during the training over another batch. This process is repeated until running through the predesignated amount of epochs.*

Empirically, the technique of dropout can reduce overfitting. One of the explanations is that the process of dropping different sets of neurons and training with different temporary networks can help avoid the overfitting effect in each of the large number of networks, which resembles a powerful averaging scheme. Another

heuristic explanation is that dropout can prevent the co-adaptations among neurons and try to avoid the presence of any individual neuron to become too significant, which helps improve the robustness of the whole neural network.

Furthermore, to artificially expand the training data set is also a successful technique to reduce overfitting. While in general it can be relatively expensive to collect a larger amount of training data points, there are some approaches helping generate nontrivial new training data based on the originally collected training data. For example, in the image recognition problem, the existing images can be translated, twisted or rotated to some limited extent to artificially generate new training data.

6.3.3 Unstable Gradient Problem

The unstable gradient problem is a generalization and extension of the aforementioned learning slowdown problem, which can be categorized into the vanishing and exploding gradient problem.

Definition 6.3.6 (Vanishing gradient problem)**.** *Vanishing gradient refers to the problem that gradients get overly small in the earlier hidden layers while the backpropagation is conducted, which causes the parameter updating in the earlier layers to be too slow.*

Definition 6.3.7 (Exploding gradient problem)**.** *Exploding gradient refers to the problem that gradients increase exponentially in the earlier hidden layers while the backpropagation is conducted, which causes the parameter learning in the earlier layers to be too unstable.*

To understand the unstable gradient problem, a simple and straightforward example is give below.

Example 6.3.1. *Assume there is a network with one input layer, three hidden layers and one output layer, each of which consists of only one sigmoid neuron. We denote the weight, bias, weighted sum and outcome associated with the neuron in the l^{th} layer as w_l, b_l, z_l and a_l. Assume a small change Δb_1 is made in the bias b_1, then this will lead to a change in the final cost function, so we have*

$$\frac{\partial C}{\partial b_1} \approx \frac{\Delta C}{\Delta b_1},$$

which indicates that the changes caused by Δb_1 can be tracked in each layer in between. Firstly, Δb_1 directly causes a change in the output Δa_1,

$$\begin{aligned} \Delta a_1 &\approx \frac{\partial a_1}{\partial b_1}\Delta b_1 \\ &= \frac{\partial \sigma(w_1 a_0 + b_1)}{\partial b_1}\Delta b_1 \\ &= \sigma'(z_1)\Delta b_1, \end{aligned} \tag{6.2}$$

where $\sigma'(z_1)$ denotes $\dfrac{\partial C}{\partial b_1}$.
Then, Δa_1 causes a change in the weighted sum input to the next layer, so we have

$$\begin{aligned} \Delta z_2 &\approx \frac{\partial z_2}{\partial a_1}\Delta a_1 \\ &= \frac{\partial (w_2 a_1 + b_2)}{\partial a_1}\Delta a_1 \\ &= w_2 \Delta a_1. \end{aligned} \tag{6.3}$$

By substituting equation 6.2 into equation 6.3, we can express how the change Δz_2 is caused by the change Δb_1,

$$\Delta z_2 \approx w_2 \sigma'(z_1) \Delta b_1.$$

Similarly, the change Δz_2 will cause a change Δa_2, and Δa_2 will further cause Δz_3, and so on, until propagating through the whole network, so ΔC can be expressed as

$$\Delta C \approx \sigma'(z_1) w_2 \sigma'(z_2) w_3 \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4} \Delta b_1.$$

Thus, the gradient can be written out as

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) w_2 \sigma'(z_2) w_3 \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4}. \tag{6.4}$$

To explain why the vanishing gradient problem occurs, we can compare the gradients in different layers.

In the same fashion, we can also write out

$$\frac{\partial C}{\partial b_3} = \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4}. \tag{6.5}$$

Equation 6.4 and 6.5 share the common terms $w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4}$. Now we need to estimate the value of the terms $w_l \sigma'(z_l)$.

For sigmoid function, by definition 6.2, we know its derivative satisfies $|\sigma'(z_l)| < 1/4$. Assume a standard approach of weight initialization is implemented, in which the weights are randomly drawn from a standard normal distribution before training, so it is mostly likely that the weights satisfy $|w_l| < 1$. Therefore, the terms $w_l \sigma'(z_l)$ usually satisfy $|w_l \sigma'(z_l)| < 1$.

Thus, as $\frac{\partial C}{\partial b_1}$ has two extra $w_l \sigma'(z_l)$ terms comparing to $\frac{\partial C}{\partial b_3}$, it is likely that $\frac{\partial C}{\partial b_1}$ is a factor of 16 smaller than $\frac{\partial C}{\partial b_3}$, which displays how the gradient vanishing problem has occurred in this network.

A rigorous proof is not given here, but from the above example we can argue that the gradients in the earlier layers are built on top of those in the later layers. Depending on the activation function and weight initialization, if the extra terms in the earlier layers compared to the later layers are smaller than 1, then it is likely that the gradient can keep decreasing from layer to layer while the change is being propagated backward in the network; if the extra terms are larger than 1 instead, then the gradient can keep increasing backward in the network. The former is the so-called vanishing gradient problem, and the latter is the exploding gradient problem. We can see that the unstable gradient is a fundamental problem related to the backpropagation in the gradient-based learning for neural networks, and it can be more serious and much easier to be triggered in more complex and deeper neural networks.

One possible technique that can be implemented to mitigate the unstable gradient problem is batch normalization. It was firstly published in the paper *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* [9] by Google in 2015.

Definition 6.3.8 (Batch normalization)**.** *Batch normalization refers to a type of technique that normalizes the input* $\{x_{1,2,...,m}\}$ *to certain layers across each batch according to algorithm 4. Batch normalization can be seen as a function* $BN_{\gamma,\beta}(x_i)$ *with two parameters* γ *and* β *which are learnt through the backpropagation.*

Algorithm 4 Batch normalization algorithm [9]

1: **Obtain** $\mathcal{B} = \{x_{1,2,...,m}\}$: obtain and store m values of a specific activation x in a mini-batch of m training examples.
2: **Compute mini-batch mean**: $\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$.
3: **Compute mini-batch variance**: $\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$
4: **Normalize**: $\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$, with ϵ a constant added to the mini-batch variance for numerical stability.
5: **Scale and shift**: $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i), \forall i = 1, 2, ..., m$.
6: **Output**: $\{y_i = BN_{\gamma,\beta}(x_i), \forall i = 1, 2, ..., m\}$

A more general solution is to carefully tune the activation function so that the extra terms in the gradients of earlier layers can be quite close to 1, which can hardly happen empirically. Thus, modern deep neural networks with gradient-based learning strategy in general are suffering from the unstable gradient problem.

6.3.4 Hyperparameter Tuning

As we have discussed in the last section, various problems can be arisen during the training of neural networks. Moreover, there is a large hyperparameter space, including the structure of the architecture, width and depth of the network, activation function, cost function, size of mini-batch, number of training epochs, weight initialization, learning rate, regularization technique, regularization parameters, and so forth. Thus, the initialization of neural networks is usually very difficult, and it takes efforts to get non-trivial learning and outcomes that are better than chance or those from the baseline models. After constructing a non-trivial network, it is also challenging to improve the performance of the network and achieve hyperparameter optimization, which is a huge research topic in deep learning that has not yet been solved.

Depending on the specific learning task and neural network, the hyperparameter tuning strategy can vary a lot. Besides, it also depends on the personal preference of the practitioners. In this thesis, neural networks are applied in financial modelling. There are some heuristics rather than strict workflows to follow as suggestions while setting the hyperparameters.

As earlier mentioned, the first goal is to get non-trivial learning and outcomes. To speed up this difficult and almost random initial searching session, we can narrow

down the learning task and shorten the training process. For instance, in our case of forecasting the interest rate during 12 projection years, we can start with a simple shallow network with 4 projection years, and we then train it with a large amount of mini-batches and a small number of epochs to monitor the validation error frequently enough and to avoid the non-trivial learning to last for too long. After this first goal is achieved, we can refine all the hyperparameters to improve the performance. In this section only the selection regarding learning rate and regularization parameter are discussed as examples.

As shown in the updating rule of gradient descent in definition 6.1.11, learning rate η decides the size of the change in parameters. If the learning rate is too large, then the change in parameters is huge per step and it is hard to specifically reach the point in the parameter space corresponding to the minimal gradient. Oppositely, if the learning rate is too small, then the change in parameters is subtle per step and it may take too long to achieve obvious gradient descent. Thus, we need to get the biggest learning rate that can still allow the gradual gradient descent and that will not cause dramatic oscillations in costs. An approach in selecting the learning rate is to get the rate of the appropriate order first and then refine it to be as large as possible.

L2 and L1 regularization are commonly implemented in neural networks to reduce overfitting, which introduces another hyperparameter, namely, regularization parameter λ. By the definition 6.3.3 and 6.3.4 of L2 and L1 regularization, we can see that λ decides how much the large weights are punished. We can select the learning rate without regularization, then tune the regularization parameter λ, and refine the learning rate again.

Besides the above basic examples regarding selecting learning rate and regularization parameter, there are more advanced techniques and many other hyperparameters to be determined. A book for hyperparameter optimization is *Neural Networks: Tricks of the Trade* [15], which is a collection of important papers in this field, including *Practical recommendations for gradient-based training of deep architectures* [1] and *Efficient BackProp* [13].

6.4 Convolutional Neural Network

There are two categories of neural networks, feedforward and recurrent neural networks. In a feedforward neural network, the output from one layer is fed as input to the next layer(s), during which the information can only be fed forward. Instead, if there are feedback loops in a network, during which the information is fed from the later layers to the earlier layers, then it is a recurrent neural network.

The most basic feedforward network is fully-connected neural network.

Definition 6.4.1 (Fully-connected neural network)**.** *A fully-connected neural network is a feedforward network in which every two neurons in every two adjacent layers are connected with each other.*

A more complicated feedforward network is convolutional neural network (CNN), in which there are three essential ideas, namely, local receptive fields, shared weights and pooling. Unlike in the fully-connected neuron network, the input layer can have

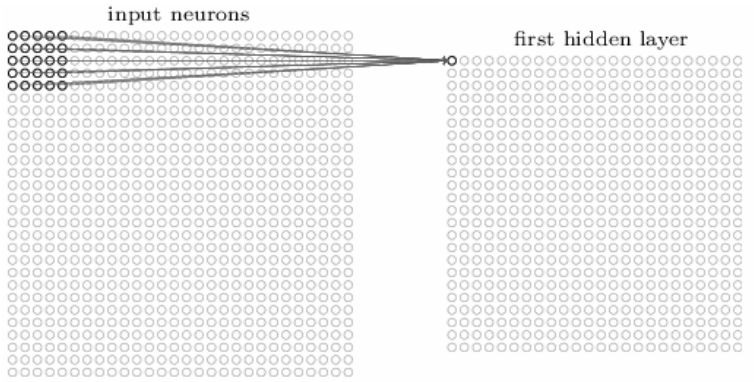
multi-dimensions, and a hidden neuron is connected with a small region of adjacent neurons.

Definition 6.4.2 (Local receptive field)**.** *The local receptive field of a hidden neuron is the small region of input neurons that are connected with the hidden neuron.*

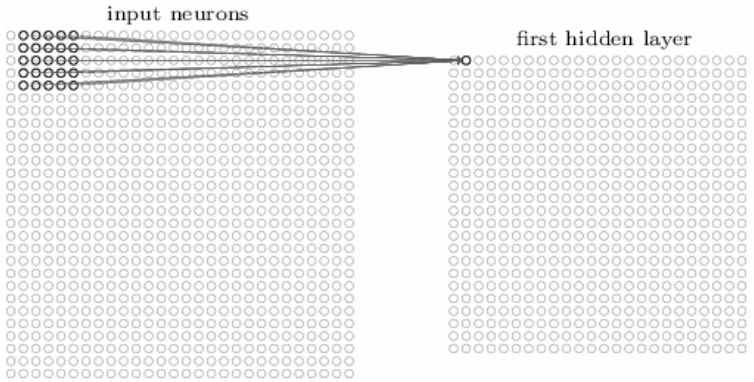
The size of the local receptive field is one hyperparameter that needs to be considered. The local receptive field can be slid along the input layer to be connected to different hidden neurons in the next layer, which introduces another hyperparameter.

Definition 6.4.3 (Stride length)**.** *Stride length is the number of input neurons that the local receptive field can be slid along the input layer per step.*

Example 6.4.1. *We take a basic example in image recognition. The input layer consists of* 28×28 *neurons to hold the input information of an image with* 28×28 *pixels. Assume the first hidden layer is a convolutional layer. The size of the local receptive field is set as* 5×5 *and the stride length is 1. As shown in figure 6.6, the local receptive field can be slid across the input neurons and is connected to a hidden neuron per step. In this case, the corresponding first hidden layer consists of* 24×24 *hidden neurons.*



(A) The first local receptive field is connected with the first hidden neuron in the next layer.



(B) The first local receptive field is slid for one stride length to form the next local receptive field which is connected with the second hidden neuron in the next layer.

FIGURE 6.6: An example of a CNN with 5×5 local receptive fields and a stride length of 1 [16].

Unlike in the fully-connected neural network where the weights and bias associated with neurons are usually different from each other, in a CNN all the hidden neurons share the same set of weights and bias.

Definition 6.4.4 (Shared weights and bias)**.** *Assume the l^{th} layer is a convolutional layer in a neural network. Assume the size of the local receptive field is $M \times M$ and the stride length is 1. Then the output $a^l_{j,k}$ from the j,kth neural in the l^{th} layer can be expressed as*

$$a^l_{j,k} = \sigma\Big(b^l + \sum_{n=0}^{M}\sum_{m=0}^{M} w^l_{n,m} a^{l-1}_{j+n,k+m}\Big),$$

where σ denotes the activation function associated with the j,kth neural in the l^{th} layer, the set of weights $w^l_{n,m}$ is called the shared weights, and b^l is the shared bias. The shared weights and bias together define a mapping from the two layers, which is known as the feature map, filter or kernel.

Just like in the human cognitive neural system, there can also be multiple channels processing information separately in the artificial neural system, so the number of channels is also a hyperparameter to be determined. For instance, in a learning task of colorful image recognition, there can be three channels of input corresponding to three numbers of pixels. Besides the input layer, the hidden layers can also consist of multiple channels, which requires multiple feature maps. Since the information output from a convolutional layer can be complex, a pooling layer is usually implemented right after the convolutional layer. There are many types of pooling procedures, among which the max-pooling and L2 pooling are introduced below.

Definition 6.4.5 (Max-pooling)**.** *The $M \times M$ max-pooling procedure outputs the maximum activation in each of the $M \times M$ regions in the previous layer.*

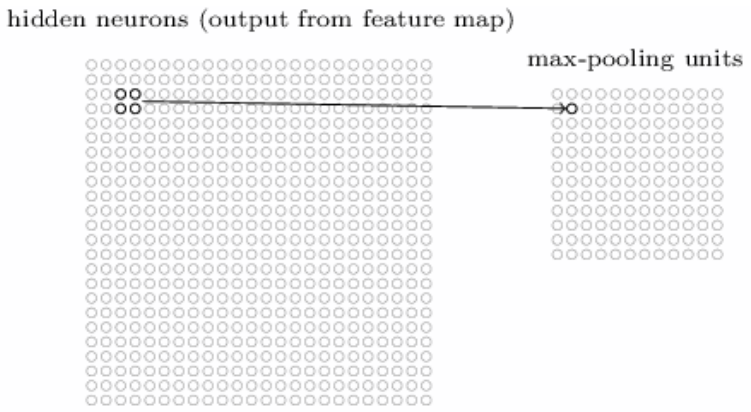


FIGURE 6.7: An example of the 2×2 max-pooling procedure [16].

Definition 6.4.6 (Average-pooling)**.** *The average pooling procedure outputs the average activation in each of the $M \times M$ regions in the previous layer.*

Convolutional layers and pooling layers together form a convolutional neural network. The major characteristic of the CNN is to incorporate the spatial structure of the input information. By implementing pooling layers as well as shared weights and bias, the complexity of the network is limited, which helps speed up the training session and reduce the overfitting impact.

However, sometimes it can be helpful to preserve the size of the input data, so the technique of zero padding as described in definition 6.4.7 is implemented.

Definition 6.4.7 (Zero padding). *Zero padding is the technique that adds 0s to the end or the surrounding of the output data of the current layer so that the output data can be of the same size as the input data that is input into this current layer.*

In practice, besides the 2D or 3D CNN applied to attack the image recognition problem, 1D CNN is also widely used to forecast time series, which is a simple variation of the 2D CNN illustrated in the above examples.

6.5 Recurrent Neural Network

Besides the above feedforward neural networks, we have also mentioned of the recurrent neural networks (RNNs). In this chapter, we introduce this type of neural networks and especially the long short-term memory neural networks (LSTMs).

Definition 6.5.1 (Recurrent Neural Network). *The recurrent neural network (RNN) is a category of neural networks in which there are recurrent connections between hidden neurons so that an internal state can be setup to store memory and the dynamic behavior over time can be exhibited.*

A simplistic illustration of RNN is shown in figure 6.8. The time-dependent input x is passed forward to the hidden neurons and incorporated into the values in the hidden neurons, namely, the state h, and the state h can be further passed forward through time. In short, in an RNN the state depends on both the input and the previous state, i.e., $h^{(t)} = f(h^{(t-1)}, x^{(t)})$. The final output from the RNN can then be computed based on the state, which is not shown in the figure here.

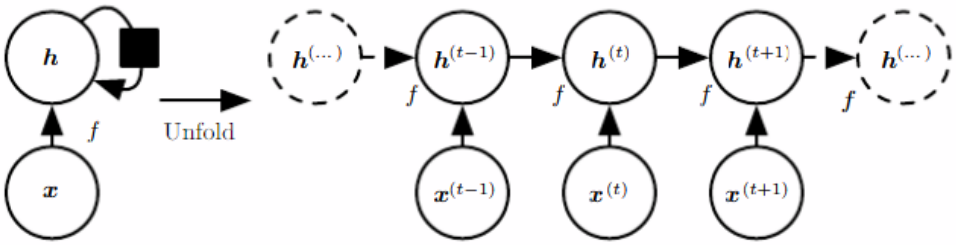


FIGURE 6.8: An illustration of RNN without the output neurons [5].

Unlike in the feedforward neural networks, in RNNs the outputs are determined not only by the activations from previous layers, but also by the activations from earlier times [16]. Due to this temporal characteristic, RNNs are commonly implemented to conduct learning tasks related to time series, such as machine translation, speech recognition and time-series data analysis.

However, one major problem with RNN is the aforementioned vanishing gradient problem, because the gradient needs to be propagated backwards not only in spatial layers but also in temporal steps in an RNN. This problem is extremely serious if there is a long-term dependency of the output on the input information, i.e., a large gap in time steps between the output and the relevant input information. As we have argued before, even though theoretically it is possible to carefully tune the hyperparameters to fall into a small range so that the the gradient can backpropagate without changing its scales exponentially, in practice it is hardly feasible to solve the unstable gradient problem in this way. This drawback is discussed in an early publication by Hochreiter in 1991 [7].

To effectively reduce the problem of unstable gradient in RNNs, Hochreiter and his professor Schmidhuber designed a variation of RNNs, the long short-term memory neural network (LSTM), in 1997 [8], which was improved by many researchers later. In the recent few years, the LSTM network has become an essential element in many breakthrough products such as Google Translate and Siri from Apple. As one of the most successful and widely-applied RNNs, the LSTM is chosen to be implemented in this thesis research project, and is also the focus of this chapter.

6.5.1 Long Short-Term Memory Neural Network

In the LSTM, the core neurons are known as LSTM cells, which have an internal recurrence and can store an internal state variable s_t corresponding to each time step t as a function of the previous state s_{t-1}. Besides, there are also three categories of hidden layers which are called gates to control the flow of data, that is, input gates, output gates and forget gates. A simple diagram of a LSTM block is illustrated in figure 6.9, in which $\times$ and $+$ denotes the operations of Hadamard product and summation, respectively.

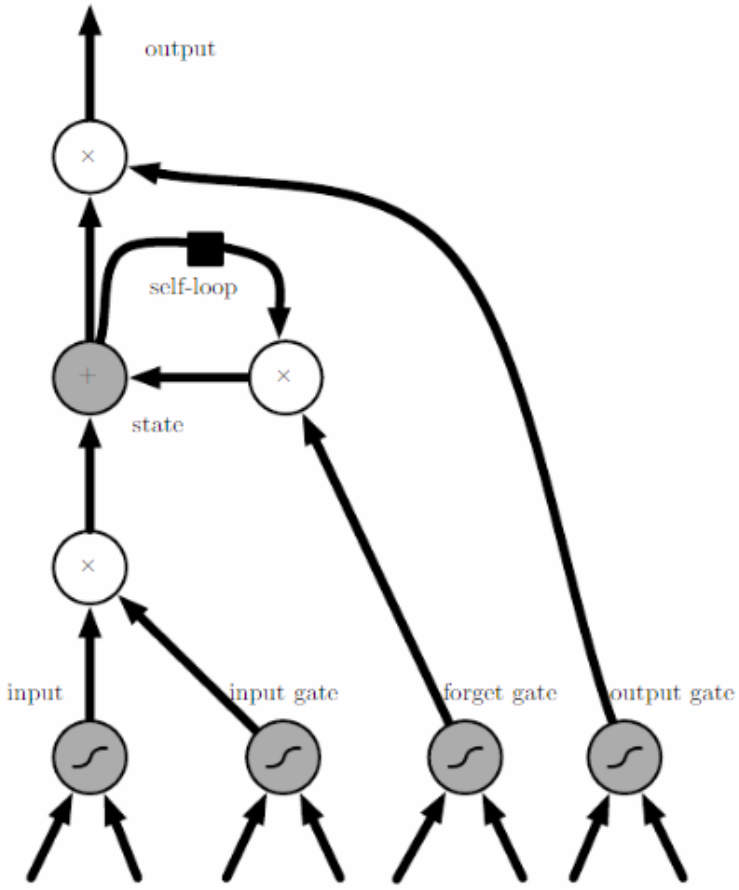


FIGURE 6.9: A diagram of LSTM neural network [5].

There are three main stages in LSTM networks, namely, the stage of input, the stage of the forget gate and state loop, and the stage of output gate [18, 5].

Assume a time series x_t is the input of the LSTM neural network. The stage of input consists of two hidden layers, the layer to scale the input information and the layer

of input gate. Firstly, the input is processed by the scaling layer to generate the output with values in $(0, 1)$,

$$g = \tanh(b^g + x_t U^g + h_{t-1} V^g),$$

where h_{t-1} is the final output from the whole LSTM network for time step $t-1$, b^g is the bias associated with the input-squashing neurons, and U^g and V^g denote the weight regarding input and weight regarding the previous final output, respectively.

The input data is also processed by the input gate to generate the output

$$i = \sigma(b^i + x_t U^i + h_{t-1} V^i),$$

where b^i is the bias associated with the input gate, and U^i and V^i denote the weight regarding input and weight regarding the previous final output, respectively.

The output from the first stage is the Hadamard product of the output g and i, that is, $g \odot i$. This stage decides what new information should be stored in the cell state.

In the second stage, the input is processed by the forget gate to generate the output with values in $(0, 1)$,

$$f = \sigma(b^f + x_t U^f + h_{t-1} V^f),$$

where b^f is the bias associated with the forget gate, and U^f and V^f denote the weight regarding input and weight regarding the previous final output, respectively. The hidden layer of forget gate takes the current input and the previous output in order to control what information that is already stored in the cell state should be forgotten.

Then the output from this stage is the updated value of the state variable

$$s_t = s_{t-1} \odot f + g \odot i.$$

As indicated in the above equations, if f is close to 0, then the previous state can be forgotten at the current time step; if f is close 1, then the previous state can be stored and passed through to set the new state.

In the third stage, the input is processed by the output gate to generate the output with values in $(0, 1)$,

$$o = \sigma(b^o + x_t U^o + h_{t-1} V^o),$$

where b^o is the bias associated with the output gate, and U^o and V^o denote the weight regarding input and weight regarding the previous final output, respectively. The layer of output gate determines what information should be output.

In the end, the final output from the whole LSTM network at time t is

$$h_t = \tanh(s_t) \odot o,$$

which is a filtered and rescaled version of the information stored in the current cell state.

The structure of LSTM ensures that the amount of the information in the system can be controlled and the problem of long-term dependencies can be resolved. Thus, LSTM is capable of remembering information and dealing with data in a long time duration with little unstable gradient problem.

Chapter 7

Methodology and Deep Neural Network Models

In this chapter, the methodology of this thesis and the structure of the model consisting of deep neural networks are elaborated.

7.1 Methodology

7.1.1 Overview of Models

The goal of this thesis is to construct a model of deep neural networks to improve the financial market model in URM. Thus, there are three categories of models implemented in Python in this research, namely, the persistence forecasting model, the financial market model applied in URM, and the model consisting of deep neural networks, as shown in figure 7.1. The output of all three models is then provided to an annuity generator so that the pension entitlement development curves can be computed and compared.

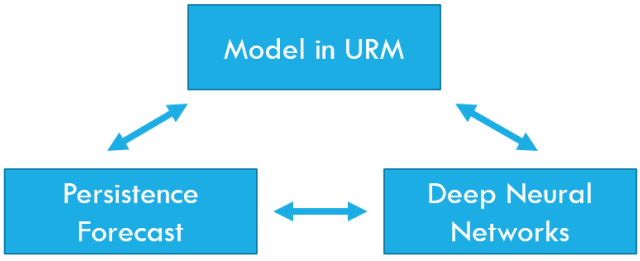


FIGURE 7.1: There are three categories of models implemented in this research, each of which is compared with the other two for the evaluation of performance.

The persistence forecasting model simply takes the last observed historical data point as the constant forecast for the future, which serves as a naive baseline model to check the triviality of the model in URM and the deep neural networks. The constant monthly predictions can then be annualized and compared with the other two models.

The second model we implement is the financial market model of URM as discussed in chapter 5. The estimated values of the time-independent parameters used in this thesis are the first set of results presented in table 5.1, which is obtained in the CPB background paper [3]. However, we need to also estimate the values of four time-dependent parameters, that is, the initial values of the state variables X_1 and X_2 as

well as the values of the parameters $\Lambda_{0(1)}$ and $\Lambda_{0(2)}$ need to be calibrated according to the financial market at the time right before the starting point of the simulation, which is discussed in appendix A. The URM model can directly predict the annual long-term interest rates, annual bond returns and annual stock returns, from which the annuity development curve can be indirectly computed. Specifically, three representative scenarios are studied, that is, the three scenarios corresponding to the 95th-, 50th- and 5th-percentile final annuities are studied as the optimistic, expected and pessimistic scenarios [12].

The last model we implement is the deep neural network model. The architecture of the deep neural networks consists of both convolutional and LSTM layers, which is elaborated in section 8.2. The deep neural networks can directly make monthly predictions of interest rates and stock prices, from which the annual predictions of interest rates, bond returns and stock returns as well as annuity development curves can be computed.

7.1.2 Processing of Data

The monthly historical data of 10-year treasury rates and stock prices is split into three data sets, namely, the training set, validation set and test set, which correspond to the time periods of the training period, validation period and test period, respectively. The test set is used for the final evaluation, so it is stored aside and not applied until all the models are finalized and the experiment is at the last stage. Instead, the validation set is used as the substitution of the test set while training to keep monitoring the performance of the models and to help guide the refinement of the models. The data in the training period is further prepared into small samples, each of which includes a list of input data and the corresponding label. The training data is used to obtain the persistence forecast during the validation period, to estimate the initial values of the parameters for the model in URM, and to train the deep neural networks.

7.1.3 Pipeline

The first step is to compare the monthly predictions directly generated by DNNs and the persistence forecasting model in order to train the DNNs to outperform the persistence model, which is elaborated as follows. The monthly prediction of interest rates and stock prices generated by the persistence forecasting model are compared with the real-world data, and the rooted mean squared errors (RMSEs) are computed. These RMSEs are set as the baselines. In the same way, the RMSEs regarding the predictions of monthly interest rates and monthly stock prices generated by the DNNs can also be computed. The DNNs can only be saved if their RMSEs are lower than the baselines, otherwise the architecture along with the trained parameters are abandoned. By training the deep neural networks to get the RMSEs lower than those from the persistence forecasting model and applying regularization techniques to reduce overfitting, we can obtain a DNN model that is likely to outperform the baseline model in the test period.

The second step is to compare the annual predictions generated by DNNs and the model in URM. We compute the annual interest rates, annual bond returns and annual stock returns based on the aforementioned monthly predictions from DNNs, compare them with the real-world data, and compute the RMSEs. Then, we generate the annual predictions from the model in URM, comprare them with the real-world data, and compute the RMSEs. If the RMSEs from DNNs are significantly lower than those from the model in URM, then it is likely that the DNNs can also outperform the model in URM during the test period. If the RMSEs from DNNs are not lower than those from the model in URM, then we need to return to step one, lower the baseline, i.e., lower the values of the RMSEs that decide if the DNNs can be saved or discarded, so that we can get another set of DNNs. We repeat these two steps until we obtain a set of DNNs that is likely to outperform the model in URM during the test period. If this never happens, then we take the models that can generate the least RMSEs that we can obtain.

In the test stage of the experiment, we use the saved models to generate predictions during the test period, compute RMSEs by comparing with the real-world data, and evaluate if the models of DNNs outperform the model in URM.

7.2 Deep Neural Network Models

The deep neural network models in this research are univariate models with monthly data as input, where 'univariate' refers that the interest rates and stock prices are separately predicted in two classes of models. However, during the research, univariate and multivariate models with annual data as input have also been considered whereas not chosen in the end, which is briefly explained in this section.

7.2.1 Annual Input Data and Multivariate Deep Neural Networks

Even though it is easy to access the annual historical data of interest rates, bond returns and stock returns, and it is convenient to generate annual predictions so as to directly compare with those predicted by the model in URM without annulizing any quantity, the annual historical data is rather sparse. Empirically, the more training data points there are, the better DNNs can be trained [16], so in the end the monthly input data is used to remarkably enlarge the data set.

During the research, a simple multivariate model with annual input data is constructed and compared with a univariate model with annual input data. While it could be possible to take advantage of the correlations among several input time series, it is challenging for the neural network to distinguish different variables among the multiple input time series, that is, the output predictions of different variables end up with having similar values and being in the same order of magnitude.

Hence, the annual input data is not adopted, nor are the multivarate models.

7.2.2 Univariate Deep Neural Networks with Monthly Input Data

The model of univariate DNNs with monthly input data consists of two separate DNN models, one for predicting interest rates and the other for stock prices. As

aforementioned, the annual predictions of interest rates, bond returns and stock returns can be computed based on the monthly predictions and are input into an annuity generator to generate the annuity development curves, which is illustrated in figure 7.2. The architectures and hyperparameters of the DNNs for interest rates and for stock prices are reported in the next chapter.

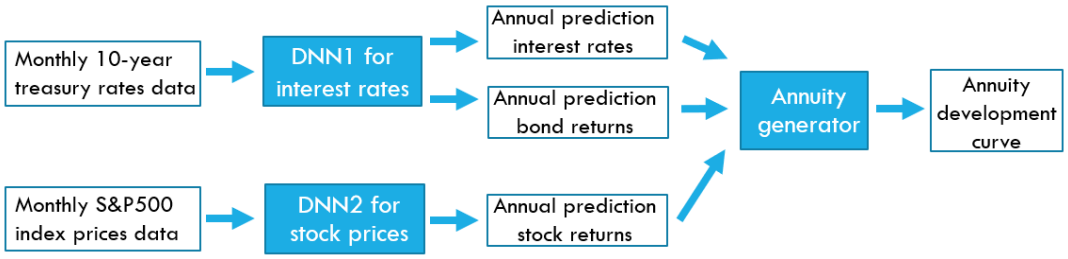


FIGURE 7.2: The basic structure of the model of univariate DNNs with monthly input data.

Chapter 8

Experimental Setup and Results

In this chapter the experimental setup and results are reported. In section 8.1, the experimental setup including the historical data and other assumptions are elaborated. In section 8.2, the architectures and hyperparameters of the DNNs for interest rates and for stock prices are reported. In section 8.3, the experimental results are analyzed qualitatively and quantitatively.

8.1 Experimental Setup

8.1.1 Data

In this thesis, the monthly U.S. financial data during 1958 to 2018 is applied because of the accessibility and representability. The interest rates are the 10-year treasury constant maturity rate published by the Federal Reserve Bank of St. Louis[1], based on which the bond returns can be computed. The stock prices and stock returns are computed based on the close prices of the S&P 500 index published by Yahoo Finance[2]. The initial values of several parameters are estimated based on the yield curves of the US treasury collected from the website of the Federal Reserve Bank of St. Louis[3].

The length of the validation period is simply set as the same as that of the test period in this thesis. As the length of the whole studied period is 61 years, we take the length of the test and validation periods to be around 10 years. If the length of the test and validation periods is less than 10 years, then this is not a long-term horizon that pension funds should have. Instead, if the length of the test and validation periods is taken to be much more than 10 years, then the training set will be much less than 2/3 of the whole data set, which may cause an insufficient training set. In this thesis, the length of the test and validation periods is set to be 12 years, because in this way the special case of the financial crisis in 2008 can be included in the test period, which helps better reflect the general performance regarding prediction of each model.

Both the real-world data and the output from the DNN models are monthly time series of 10-year treasury rates and stock prices. To compare them with the annual predictions generated by the model in URM, we need to conduct the following conversions for the real-world data and predictions from DNN models. Assume the

[1] https://fred.stlouisfed.org/series/GS10

[2] https://finance.yahoo.com/quote/%5EGSPC/history?period1=-378694800&period2=1546297200&interval=1mo&filter=history&frequency=1mo

[3] https://fred.stlouisfed.org/categories/115

monthly data of annual interest rates during year t is $\{m_1, m_2, ..., m_{12}\}$, then the annual interest rate $y(t)$ for this year can be estimated by

$$y(t) = (1 + m_1)^{\frac{1}{12}}(1 + m_2)^{\frac{1}{12}}...(1 + m_{12})^{\frac{1}{12}} - 1.$$

Then, we can approximate the annual bond returns of the 10-year treasury by

$$\frac{P(t_{i+1}) - P(t_i)}{P(t_i)} \approx y(t_i)\Delta t + \tau(y(t_{i+1}) - y(t_i)),$$

where $P(t)$ denotes the nominal bond prices of the 10-year treasury at year t, τ denotes the duration to maturity which is taken as 10 in the case of 10-year treasury, and $\Delta t = t_{i+1} - t_i$ is taken as 1 in our case for annual bond returns.

Regarding the data for stock market, assume the monthly data of stock prices during year t is $\{n_1, n_2, ..., n_{12}\}$ and the stock price during the last month before year t is n_0, then the annual stock return $z(t)$ for this year can be computed by

$$z(t) = \frac{n_1}{n_0}\frac{n_2}{n_1}...\frac{n_{12}}{n_{11}} - 1 = \frac{n_{12}}{n_0} - 1.$$

8.1.2 Annuity Assumptions

We assume the premium is invested in two categories, stock market and 10-year bond. The portfolio is rebalanced every year to keep the maturity of the bond constant, and every year the percentages of the capital invested to the stock market and to the 10-year bond are both set as 50%. Assume the starting capital in the pension account of the participant is 100,000 euro in total, and the defined contribution is 10,000 euro per year.

For simplicity, assume the first half of the test and validation periods is devoted to the pre-retirement investment, the other half is the period of getting pension entitlements after retirement, and the age of retirement is 65. In our case that the length of the test/validation period is set as 12 years, we assume that the starting point of the simulation for the participant is at the age of $65 - 12/2 = 59$, and the period during which the participant retires and can obtain pension annuity is from the age of 65 to $65 + 12/2 = 71$.

8.1.3 Software and Hardware

In this thesis all the models are implemented in Python. Specifically, the DNNs are built with the high-level neural networks library of Keras running on top of the low-level open source library TensorFlow, and the DNNs are trained in Google Colaboratory using GPU as the hardware accelerator.

8.2 Hyperparameters of DNNs

8.2.1 DNN for Interest Rates

The architecture, width and associated activation functions in the DNN for interest rates are listed in table 8.1.

TABLE 8.1: Some Hyperparameters in the DNN for Interest Rates

Layers	Type of Layer	Number of Neurons	Activation Function
1	Input Layer	12	N/A
2	Batch Normalization	N/A	N/A
3	Dense	64	ReLU[4]
4	Conv1D	704	Tanh
5	Average Pooling 1D	N/A	ReLU
6	Flatten	N/A	N/A
7	Repeat Vector	N/A	N/A
8	LSTM	128	ReLU
9	Dense	64	ReLU
10	Dense	1	ReLU
11	Flatten	1	N/A

Besides, in the one-dimensional convolutional layer, the number of the filters is set to be 64, the size of the filter is set to be 2, the stride length is set to be 1, and zero padding is applied. In the average pooling layer, the pool size is set to be 2[5]. In the LSTM layer, the dropout technique is implemented with the dropout probability set to be 0.5. While training this DNN, the number of epochs is set to be 100, and the batch size is set to be 6. The method of optimization that is implemented is Adam, with the commonly-applied default setting $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$.
The DNN for interest rates is summarized in table 8.2 and visualized in figure 8.1. The output shape refers to the size of the matrices.

TABLE 8.2: Summary of the DNN for Interest Rates

Layers	Type of Layer	Output Shape	Number of parameters
1	Input Layer	(12,1)	0
2	Batch Normalization	(12,1)	4
3	Dense	(12,64)	128
4	Conv1D	(11,64)	8256
5	AveragePooling1D	(5,64)	0
6	Flatten	(320)	0
7	RepeatVector	(2, 320)	0
8	LSTM	(2, 128)	229888
9	Dense	(2, 64)	8256
10	Dense	(2,1)	65
11	Flatten	(2)	0

[4]ReLU refers to the rectified linear neuron, as defined in definition 6.1.6.
[5]For the definitions of the terms mentioned in this part, please refer to section 6.4.

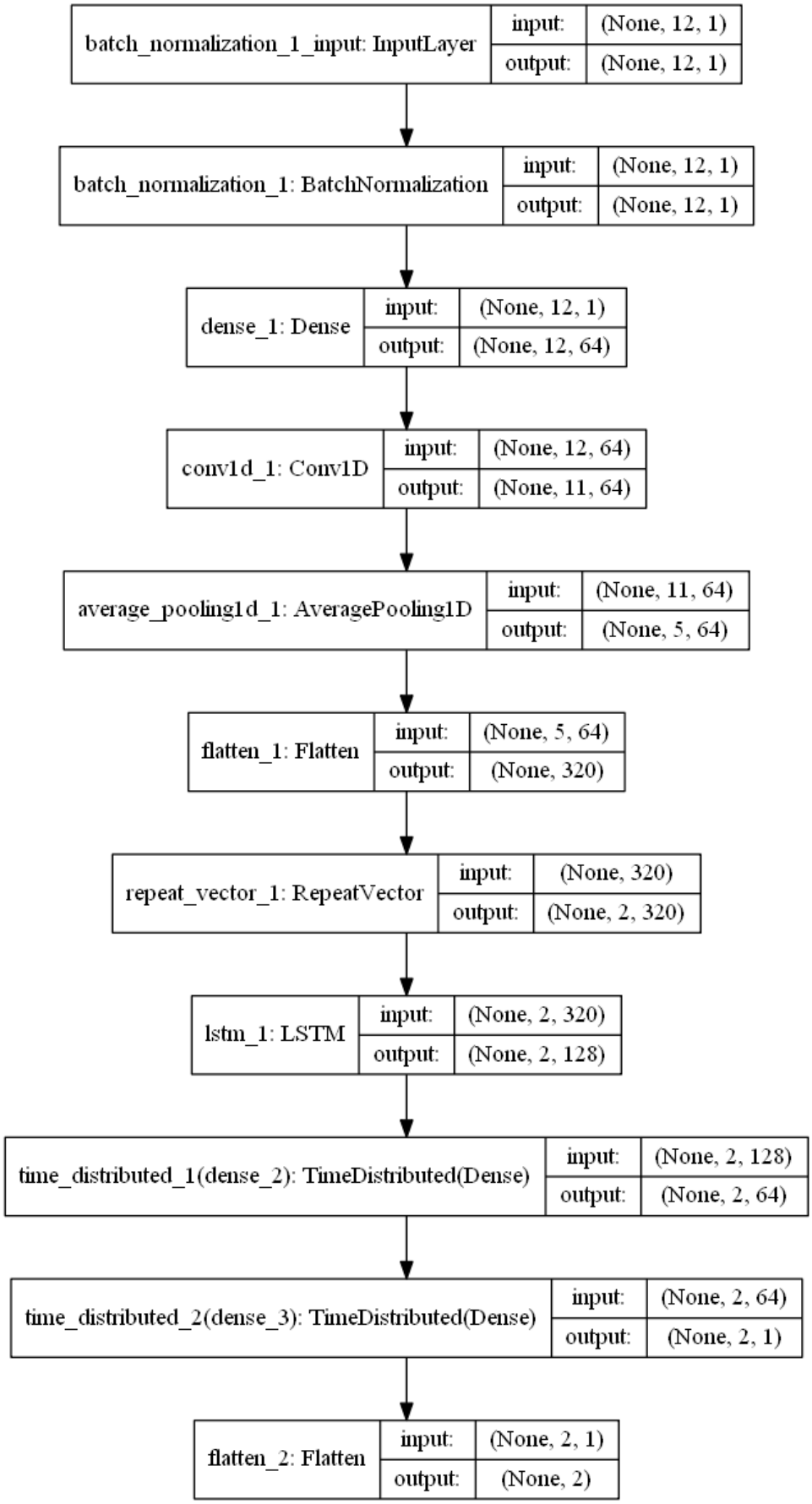


FIGURE 8.1: The architecture of the DNN for interest rates. The tuple for the input and output size represents (width, height, depth) or (width, height) when there are only two entries. In our case of one-dimensional time series, width is None.

8.2.2 DNN for Stock Prices

The architecture, width and associated activation functions in the DNN for stock prices are listed in table 8.3.

TABLE 8.3: Some Hyperparameters in the DNN for Stock Prices

Layers	Type of Layer	Number of Neurons	Activation Function
1	Input Layer	12	N/A
2	Dense	64	ReLU
3	Batch Normalization	N/A	N/A
4	Conv1D	704	ReLU
5	Average Pooling 1D	N/A	ReLU
6	Flatten	N/A	N/A
7	Repeat Vector	N/A	N/A
8	LSTM	128	ReLU
9	Dense	64	ReLU
10	Dense	1	ReLU
11	Flatten	1	N/A

Besides, in the one-dimensional convolutional layer, the number of the filters is set to be 64, the size of the filters is set to be 2, the stride length is set to be 1, and the zero padding is applied. In the average pooling layer, the pool size is set to be 2. In the LSTM layer, the dropout technique is implemented with the dropout probability set to be 0.5. While training the DNN, the number of epochs is set to be 50, and the batch size is set to be 6. The method of optimization that is implemented is Adam, with the commonly-applied default setting $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

The DNN for stock prices is summarized in table 8.4 and visualized in figure 8.2.

TABLE 8.4: Summary of the DNN for Stock Prices

Layers	Type of Layer	Output Shape	Number of parameters
1	Input Layer	(12,1)	0
2	Dense	(12,64)	128
3	Batch Normalization	(12,64)	256
4	Conv1D	(11,64)	8256
5	AveragePooling1D	(5,64)	0
6	Flatten	(320)	0
7	RepeatVector	(2, 320)	0
8	LSTM	(2, 128)	229888
9	Dense	(2, 64)	8256
10	Dense	(2,1)	65
11	Flatten	(2)	0

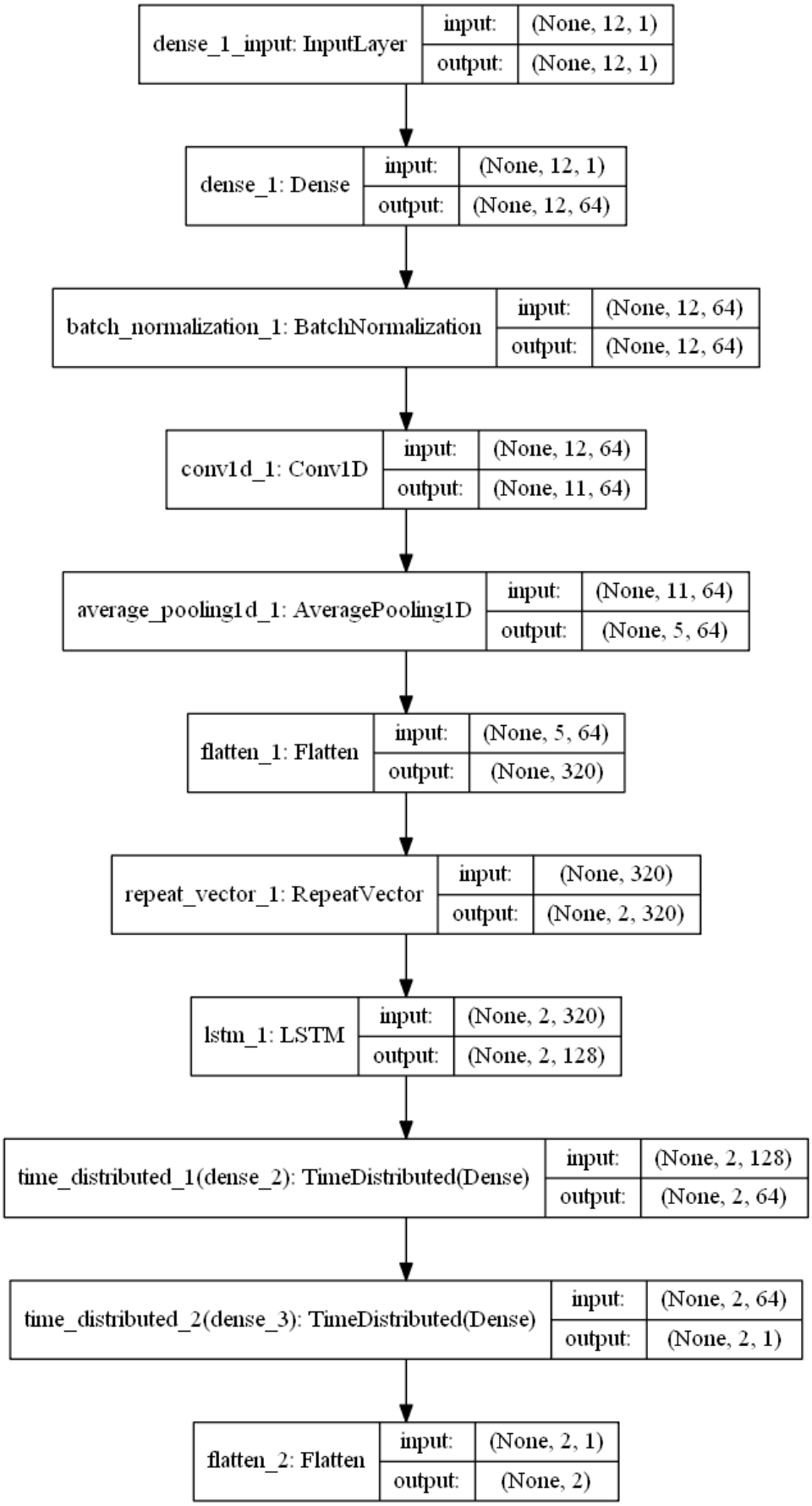


FIGURE 8.2: The architecture of the DNN for stock prices. The tuple for the input and output size represents (width, height, depth) or (width, height) when there are only two entries. In our case of one-dimensional time series, width is None.

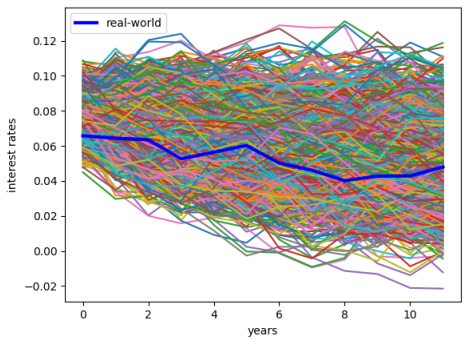
8.3 Predictions

In this section, the results from the persistence forecasting model, the model in URM and the DNN models are presented.

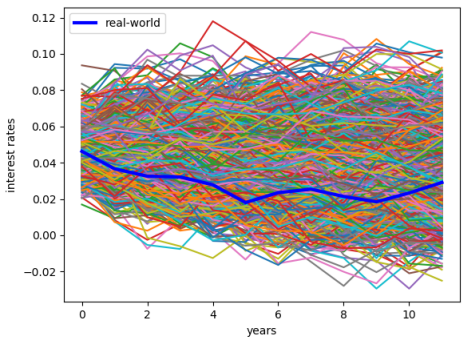
The results are plotted for both the validation period and the test period, starting from the first year in the corresponding period without showing the initial values. That is, the starting point of each quantity is the last historical data point before the validation or test period, so it is not shown in the figure; the values at year 0 in each of the figures correspond to the first predictions at the first conjecture year of the validation or test period.

8.3.1 Results Regarding Treasury Rates

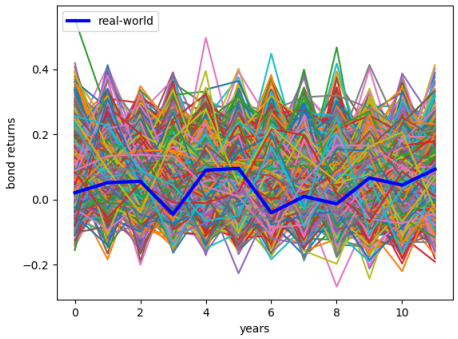
The model in URM generates 2000 scenarios of annual predictions regarding 10-year treasury rates and bond returns for both the validation and test periods. As shown in figure 8.3, the real-world data is contained by the range of the 2000 scenarios.



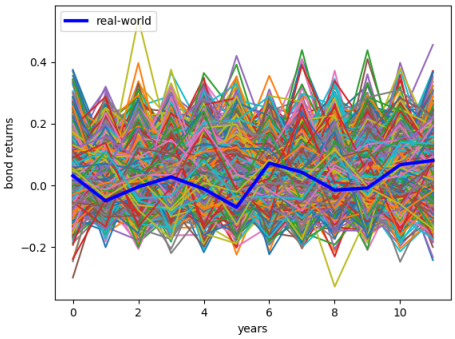
(A) Interest rates on validation period



(B) Interest rates on test period



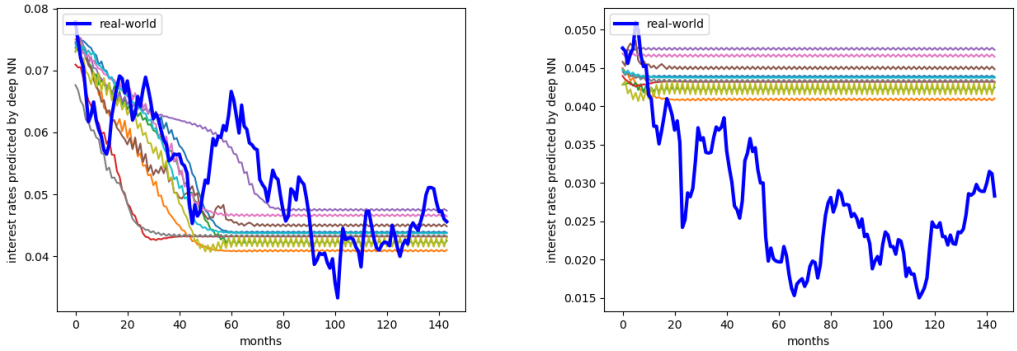
(C) Bond returns on validation period



(D) Bond returns on test period

FIGURE 8.3: The 2000 scenarios of annual predictions of the 10-year treasury rates and bond returns generated by the financial market model in URM during the validation and test periods.

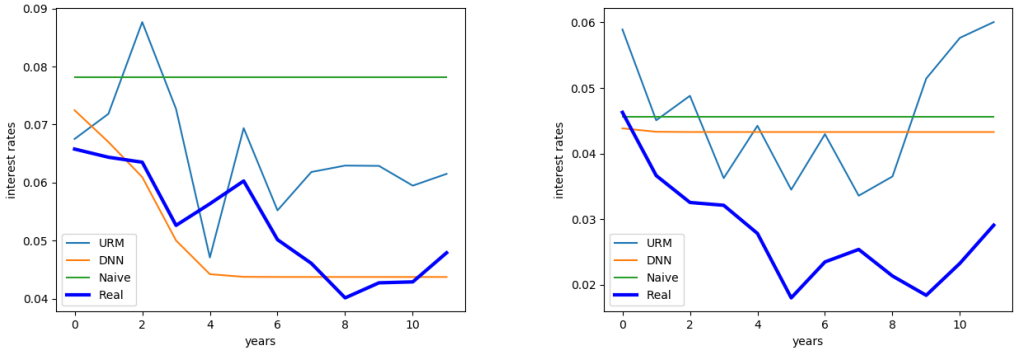
To reduce the impacts caused by peculiarities, we obtain 10 sets of different DNN models, which are separately trained with the same architecture and hyperparameters as specified in section 8.2.1. These DNN models end up obtaining different parameters, because different random seeds are implemented for initialization. Each of the 10 DNN models regarding interest rates generates one prediction of the monthly 10-year treasury rates, as shown in figure 8.4. The overall trends of the treasury rates predicted by DNNs conform with that of the real-world data during the early stage of the validation period. However, the predictions tend to fluctuate around some constant values after around 60 months during the validation period and almost throughout the whole test period. Attempts on mitigating this problem are not successful, and a possible cause would perhaps be the insufficiency of the training data, which impedes the comprehensive learning of the complicated behavior of the treasury rates and limits the ability of the model to make long-term predictions.



(A) During validation period (B) During test period

FIGURE 8.4: The monthly predictions of 10-year treasury rates generated by the DNN for interest rates during the validation and test periods.

The monthly predictions of 10-year treasury rates generated by the DNNs are converted into annual interest rates to compare with the predictions from the other models. The naive model of persistence forecast provides one deterministic prediction. For each of the 2000 scenarios generated by the model in URM, the Rooted Mean Squared Error (RMSE) is computed by comparing the predicted time series to the real-world data. The 50th percentile among all the RMSEs is selected, and the time series of annual interest rates corresponding to the 50th percentile of the RMSEs is obtained. In the same way, the 50th percentile of RMSEs and the corresponding annual interest rate are obtained from the 10 scenarios generated by the DNN models. Together with the real-world data and persistence forecast, the annual predictions of interest rates correspnding to the 50th percentile of RMSEs generated from the model in URM and the DNN models are plotted in figure 8.5.



(A) During validation period (B) During test period

FIGURE 8.5: The annual predictions of interest rates corresponding to the 50th percentile of RMSEs generated by different models.

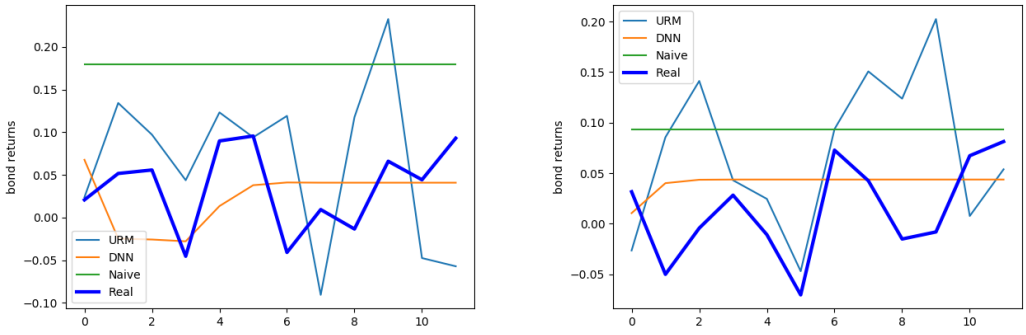
The 50th-percentile and average RMSEs regarding the predictions of annual interest rates are listed in table 8.5. A lower error of a prediction indicates a smaller deviation from the real-world data. Both the average error and the 50th-percentile error can reflect the overall performance of the corresponding model. The validation errors provide a view of how well the models fit initially, and the test errors are relatively more important regarding the final evaluation, which reflects the generalization performance and the prediction ability of the models.

According to the errors listed in the table 8.5, during the validation period, the persistence model performs the worst, the model in URM significantly outperforms the persistence model, and a further considerable improvement is seen in the DNN models in general. During the test period, the performance of the persistence forecast is relatively close to that of the model in URM, and the DNN models slightly outperform the the other two types of models.

TABLE 8.5: Validation and test errors regarding annual interest rates

Period	Type	Persistence	URM	DNN
Val Errors	Average	0.02683	0.01871	0.007578
	50th-percentile	0.02683	0.01780	0.006853
Test Errors	Average	0.01937	0.02123	0.01753
	50th-percentile	0.01937	0.01903	0.01728

From the monthly data or predictions of the 10-year treasury rates we can also compute the annual bond returns. As before, for each type of the models, the RMSE regarding each scenario of the annual bond returns can be obtained, and the 50th percentile is selected. The bond returns corresponding to the 50th-percentile error is plotted in figure 8.6.



(A) During validation period (B) During test period

FIGURE 8.6: The annual predictions of bond returns corresponding to the 50th percentile of RMSEs generated by different models.

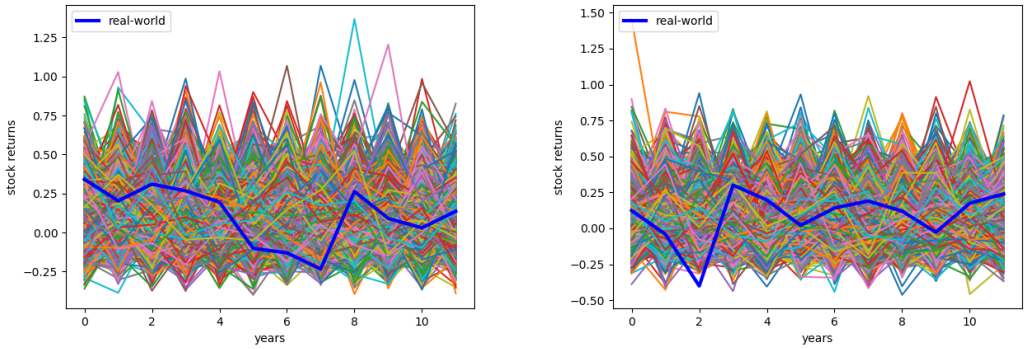
According to the errors listed in table 8.6, during the validation period, the model in URM significantly outperforms the persistence model, and the errors of the DNN models almost halve those of the URM model. During the test period, however, the persistence model slightly outperforms the URM model, and the DNN models generate much better performance than the other two in general.

TABLE 8.6: Validation and test errors regarding annual bond returns

Period	Type	Persistence	URM	DNN
Val Errors	Average	0.1513	0.1379	0.05927
	50th-percentile	0.1513	0.1036	0.05641
Test Errors	Average	0.09178	0.1034	0.05502
	50th-percentile	0.09178	0.1023	0.05481

8.3.2 Results Regarding Stock Market

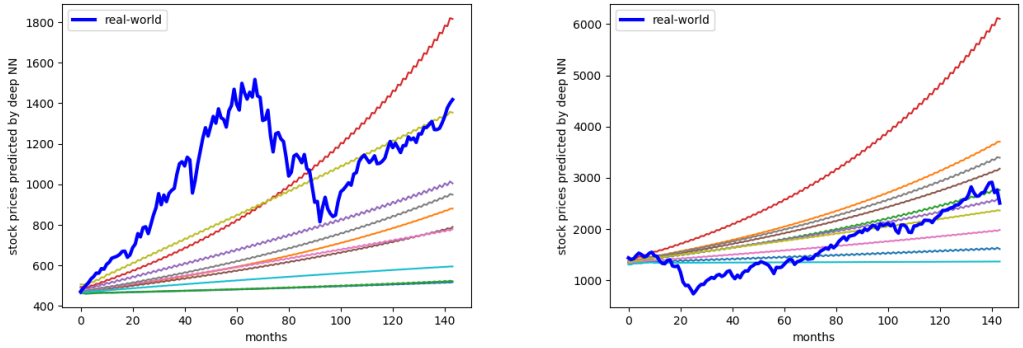
The model in URM also generates 2000 scenarios of annual predictions regarding stock returns. As shown in figure 8.7, in general, for most of the time steps, the real-world data is well contained in the range of the 2000 scenarios.



(A) During validation period (B) During test period

FIGURE 8.7: The 2000 scenarios of annual stock returns generated by the financial market model in URM during the validation and test periods.

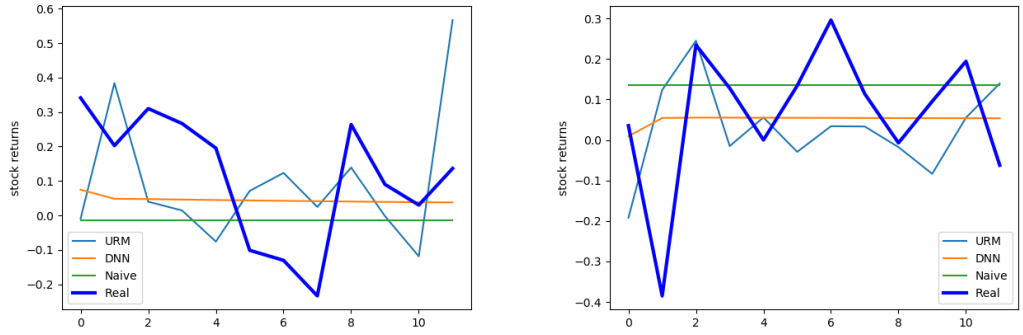
The DNN model regarding interest rates can generate predictions of monthly stock prices, as shown in figure 8.8. The predictions are in general monotone if not considering the small local fluctuations, so they cannot accurately reflect the changes in the real-world stock prices. However, the scenarios of predictions have an acceptable range of values during the validation period, and the trends of the predictions conform with that of the real-world data for most of the part during the test period.



(A) During validation period (B) During test period

FIGURE 8.8: The monthly prediction of stock prices generated by the DNN model for stock prices during the validation and test periods.

From the monthly data or predictions of stock prices we can also compute the annual stock returns. As before, for each type of the models, the RMSE regarding each scenario of the annual stock returns can be obtained, and the 50th percentile is selected. The stock returns corresponding to the 50th-percentile error is plotted in figure 8.9.



(A) During validation period (B) During test period

FIGURE 8.9: The annual prediction of stock prices corresponding to the 50th percentile of RMSEs generated by different models.

The short-term changes in stock prices are widely deemed hardly predictable [14]. According to the errors listed in table 8.7, during the validation period, the persistence model outperforms the model in URM, and the DNN models significantly outperform the persistence model in general. During the test period, the persistence model considerably outperforms the model in URM, and the DNN models slightly outperform the persistence model.

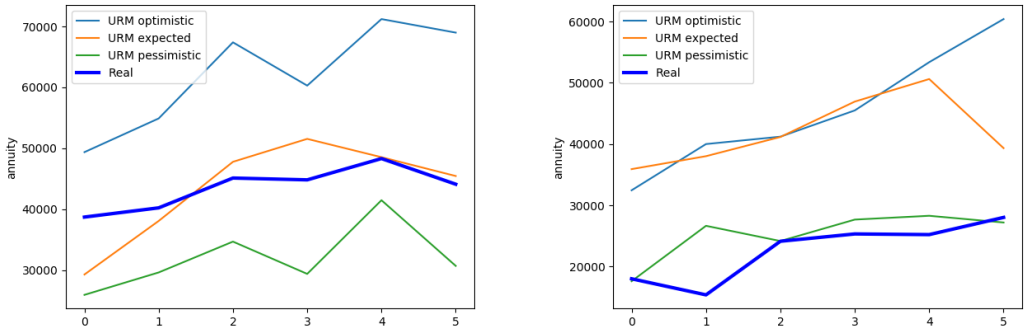
TABLE 8.7: Validation and test errors regarding annual stock returns

Period	Type	Persistence	URM	DNN
Val Errors	Average	0.2211	0.2520	0.1888
	50th-percentile	0.2211	0.2471	0.1880
Test Errors	Average	0.1828	0.2541	0.1717
	50th-percentile	0.1828	0.2775	0.1685

8.3.3 Results Regarding Pension Annuities

Based on the annual data and predictions of interest rates, bond returns and stock returns, and together with the annuity assumptions as specified in section 8.1.2, the annuity development curves can be generated for all the scenarios. According to the 95th, 50th and 5th percentiles of the final annuities, i.e., each of the annuities accumulated right before the retirement age, we can obtain three special scenarios, namely, the optimistic, expected and pessimistic scenarios.

The three special scenarios of annuity development curves that are generated by the model in URM are plotted in figure 8.10. During the validation period, the real-world data is relatively close to the expected scenario and well contained in between the optimistic and pessimistic scenarios. However, during the test period, the real-world data is slightly below the pessimistic scenario.



(A) During validation period (B) During test period

FIGURE 8.10: The annuity development curves predicted by the model in URM.

The three special scenarios of annuities generated by the URM model for the test period can be illustrated in the pension communication materials in the form of the navigation metaphor as shown in figure 8.11.

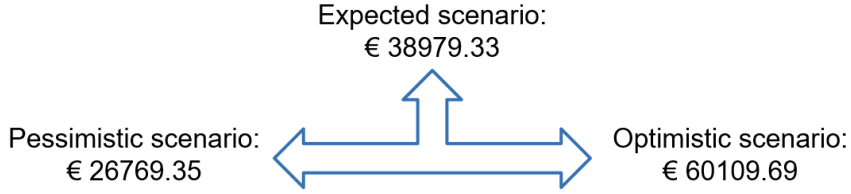
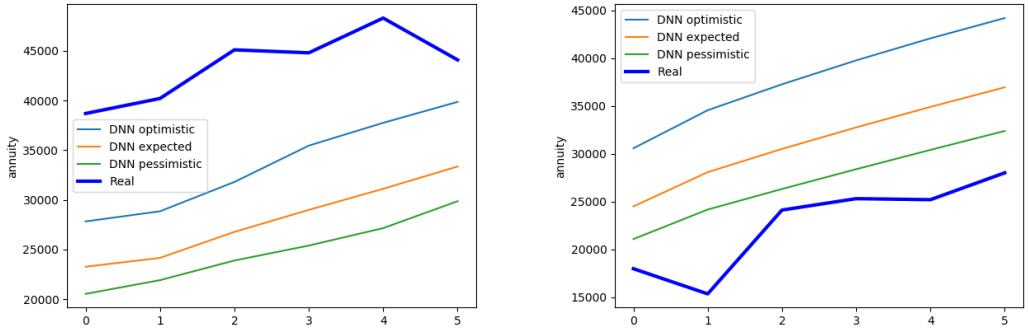


FIGURE 8.11: The three scenarios of pension annuities generated by the URM model.

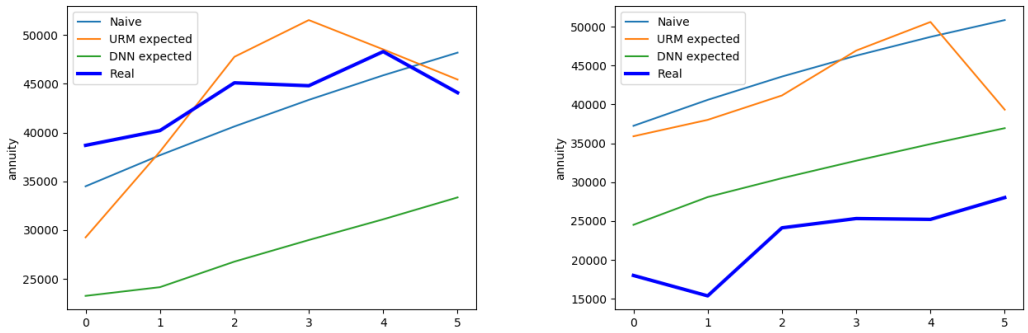
Similarly, the three special scenarios of annuity development curves that are generated by the DNN models are plotted in figure 8.12. During the validation period, the real-world annuity is significantly higher than the three scenarios of predictions, which is mainly caused by the general low stock prices predicted by the DNN models. During the test period, however, the real-world annuity is lower than the three scenarios of predictions, which is mainly caused by the almost-constant high interest rates predicted by the DNN models.



(A) During validation period (B) During test period

FIGURE 8.12: The annuity development curves predicted by the DNN models.

To visualize the differences, the expected predictions generated by different models are plotted in figure 8.13. During the validation period, the persistence forecast and the expected scenario generated by the model in URM are close to the real-world data, while the expected scenario generated by the DNN models performs the worst, almost 15,000 euro lower than the real-world scenario all the way through the annuity development curve. During the test period, the expected scenarios from all the models are higher than the real-world data, among which the expected annuity development curve generated by DNN models is the closest to the real-world data.



(A) During validation period (B) During test period

FIGURE 8.13: The annuity development curves predicted by different models.

The three special scenarios of annuities generated by the DNN model for the test period can be illustrated in the pension communication materials in the form of the navigation metaphor as shown in figure 8.14.

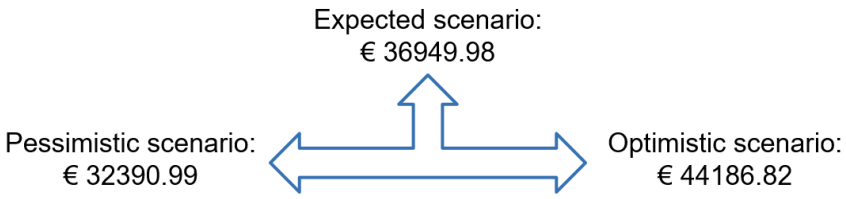


FIGURE 8.14: The three scenarios of pension annuities generated by the DNN model.

To more strictly compare and evaluate the general performance of different models, the average errors and the errors associated to the expected scenarios should be both reported, as listed in table 8.8. According to the errors, during the validation period, the persistence forecast has the best performance, and the DNN models perform the worst[6]. However, during the test period, the persistence forecast performs the worst, and DNN models considerably outperform the other two types of models regarding all of the three special scenarios and the average performance.

TABLE 8.8: Validation errors regarding annuity development curves

Period	Type	Persistence	URM	DNN
Val Errors	Average	3388.57	9728.71	15098.28
	Optimistic	3388.57	13633.01	10334.94
	Expected	3388.57	8486.95	15775.63
	Pessimistic	3388.57	13838.31	18894.57
Test Errors	Average	21978.35	13735.16	9270.02
	Optimistic	21978.35	28497.25	15564.00
	Expected	21978.35	13294.37	8888.95
	Pessimistic	21978.35	7928.38	4961.51

[6]Note that during the validation period, the DNN models in general have the lowest RMSEs regarding interest rates, bond returns and stock returns, but the DNN models in general have the highest RMSEs regarding the annuity. An explanation could be that in the persistence model and URM model the impacts of the deviations from the real-world data regarding interest rates, stock returns and bond returns cancel out with each other more than in the DNN models, so that the deviation regarding the annuity is smaller in the persistence model and URM model. For example, during the validation period, the persistence model forecasts higher bond returns than the real-world data, whereas forecasts lower stock returns than they should have been, and these two deviations cancel out with each to some extent while computing the annuity so that there is not a big deviation regarding the predicted annuity.

Chapter 9

Conclusion

In this chapter the conclusion based on the experimental results reported in chapter 8 is summarized.

In this thesis, the experiment is conducted with the historical data from 1958 to 2018, and the final performance is evaluated during the 12-year test period of 2007 to 2018. The financial market model in URM is implemented to generate 2000 scenarios of nominal variables. Additionally, we construct a model consisting of two DNNs to forecast the financial market, among which one DNN model is developed to forecast the 10-year treasury rates and the other is for the stock prices. 10 scenarios of interest rates, bond returns and stock returns can be generated by 10 sets of separately-trained DNN models. The annual predictions regarding interest rates, bond returns and stock returns generated by the model in URM, the DNN models and the persistence forecasting model are compared and evaluated, where RMSE is applied as the metric of measuring the deviation of the predicted time series from the real-world data. Accordingly, the pension entitlement development curves can be computed based on the aforementioned predictions, and three special scenarios, namely, the optimistic, expected and pessimistic scenarios are analyzed for each type of the models.

During the test period, regarding the annual predictions of interest rates, bond returns and stock returns, on average the DNN models keep outperforming both the model in URM and the persistence model. As for the annuity development curves, the average test error and the test errors of the three special scenarios generated from the DNN models are all considerably lower than those from the URM model and the persistence forecasting model, which indicates that the DNN models in general and on average provide more accurate predictions of pension annuity development curve compared to the model in URM and the persistence forecasting model. However, by figure 8.10, 8.12 and 8.13, none of the models can actually provide an expected scenario that is fairly close to the real-world data and acceptable optimistic and pessimistic scenarios that can cover the range of the real-world data.

In this thesis, the research question is to investigate the possibility of developing a model of deep neural networks to improve the URM model. Based on the current experimental settings and the considered training, validation and test periods, the conclusion is that the DNN models implemented in this thesis can outperform the URM model and provide relatively more accurate pension communication, even though the accuracy still needs to be further improved.

Chapter 10

Discussion and Future Work

In this chapter, the limitations of this thesis are discussed and possible future work is presented.

Above all, the lack of the historical data limits the learning of the DNN models. With a few decades to a century of financial market data, it can be rather challenging to provide a sufficient amount of training data and to obtain accurate long-term predictions. Additionally, it is also well-known that the financial market has complicated behaviors, which makes it even harder to fully train the DNNs with the sparse data set.

Moreover, a longer period of historical data is required to conduct a more comprehensive comparison and evaluation. In the set of the URM scenarios published by DNB for each quarter, there are 60 projection years. However, due to the insufficiency of the historical financial data, it is hardly possible to train the DNN models well and at the same time to evaluate them with a test period of 60 years.

Besides, one of the biggest concerns while implementing deep learning is regarding its accountability. While the technique of deep learning focuses on empowering machines to automatically learn the patterns and generate classifications or predictions without expertise in the relevant field or too much human interventions of manipulating the data, the complexity of the DNNs learnt by machines turns out to go beyond the understanding of human most of the time. When taking crucial tasks or making critical decisions, especially in the public sector, it can be controversial for the policy makers to implement DNNs without providing full accountability, even with persuasive results obtained by DNNs. Since the application of deep learning has been surging since that last decade, more regulations and techniques for evaluation are urgently needed in the near future [27].

One possibility of the future work is to further investigate the forecast from the perspective of financial and non-financial assumptions. In the URM model, the behaviour of the financial market is assumed to be relatively constant by using a large amount of time-independent parameters. It is promising to improve the URM model by implementing a more dynamic model with more time-dependent variables. Regarding the other assumptions that are not in term of the financial market, such as the premium policy, investment policy and the starting capital in the pension account of the participants, more detailed assumptions can be made to generate more realistic simulations.

Another direction of the future work could be investigating the possibility of constructing a skillful multivariate DNN model which takes multiple types of financial

time series as input. This thesis fails to develop such a multivariate model, but it should be very promising once it is realized, since it can take advantage of the correlations among time series, which is extra information that can be learnt from.

Last but not least, it could be possible to develop a generative model to artificially expand the financial data set. In the field of image recognition, the Generative Adversarial Networks (GANs) has already been implemented to automatically generate images [6]. However, it can be rather challenging to develop effective generative algorithms for time series, especially financial market data.

Appendix A

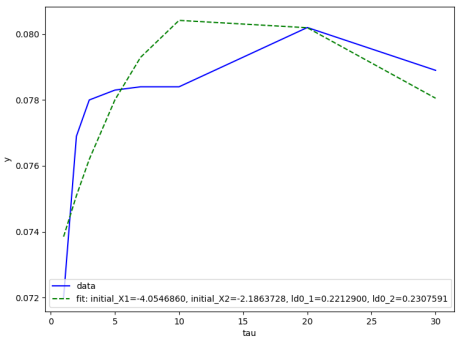
Initial Values In URM

As mentioned in section 7.1.1, the initial values of the state variables X_1 and X_2 as well as the values of the parameters $\Lambda_{0(1)}$ and $\Lambda_{0(2)}$ need to be calibrated according to the financial market at the time right before the starting point of the simulation. The estimation of the initial values follows the document [4] published by Commission Parameters.

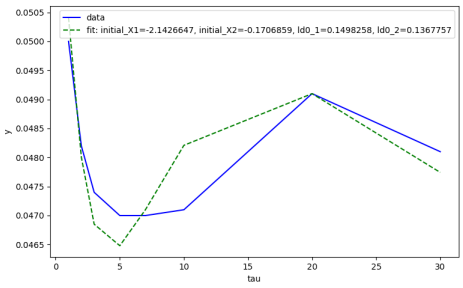
According to equation 5.10, the instantaneous nominal yield of a bond with duration τ is

$$\frac{-d \ln P(X_t, t, t+\tau)}{dt} = -\dot{A}(\tau) - \dot{B}(\tau)' X_t. \tag{A.1}$$

The yield curves of instantaneous forward rates are downloaded from the website of the European Central Bank[1], and the function A.1 is fit into the data in order to get the estimation of the initial values. In this thesis, the validation period is from 1995 to 2006, and the test period is from 2007 to 2018. Therefore, to find the initial values in URM during the validation period, we need to obtain the real-world data on the last day available before the year of 1995, which is December 30th of 1994; and to find the initial values in URM during the test period, we need to obtain the real-world data on the last day available before the year of 2007, which is December 29th of 2006. The estimation results are shown in figure A.1 and table A.1.



(A) For the validation period



(B) For the test period

FIGURE A.1: The real-world data (blue) and fitted (green) yield curves of instantaneous forward rates.

[1] https://www.ecb.europa.eu/stats/financial_markets_and_interest_rates/euro_area_yield_curves/html/index.en.html

TABLE A.1: The estimated initial values in URM

Period	Initial X_1	Initial X_2	$\Lambda_{0(1)}$	$\Lambda_{0(2)}$
Val	-4.0546860	-2.1863728	0.2212900	0.2307591
Test	-2.1426647	-0.1706859	0.1498258	0.1367757

Bibliography

[1] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[2] DNB. Rekenmethodieken voor weergave van ouderdomspensioen in scenario's.

[3] Nick Draper. A financial market model for the netherlands. 2014.

[4] GMM Gelauff, Th E Nijman, OCHM Sleijpen, and OW Steenbeek. Advies commissie parameters.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[10] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Wouter Koolmees. Regeling van de minister van sociale zaken en werkgelegenheid van 13 april 2018, nr. 2018-0000071068, tot vaststelling van de rekenmethodieken voor weergave van ouderdomspensioen in scenario's. *STAATSCOURANT Nr.22286*, 2018.

[13] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[14] Burton Gordon Malkiel and Kerin McCue. *A random walk down Wall Street*, volume 332. Norton New York, 1985.

[15] Grégoire Montavon, Geneviève Orr, and Klaus-Robert Müller. *Neural networks: tricks of the trade*, volume 7700. springer, 2012.

[16] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press USA, 2015.

[17] OECD. Pension markets in focus 2018.

[18] Christopher Olah. Understanding lstm networks, 2015.

[19] Fabian Polman and Connor Dekker. Riskco meeting slides: Uniforme rekenmethodiek (urm). 2017.

[20] Sibylle JM Reichert. The dutch pension system, an overview of the key aspects.

Brussels, Belgium: the Dutch Association of Industry-wide Pension Funds and the Dutch association of Company Pension Funds, 2014.

[21] Steven E Shreve. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer Science & Business Media, 2004.

[22] Peter JC Spreij. Measure theoretic probability. *UvA Course Notes*, 2012.

[23] SVB. Aow pension, 2019.

[24] Iris Theeuwes. Review of discount rates for the valuation of pension liabilities. *Master's Thesis Tu/e*, 2018.

[25] Rob Van den Goorbergh, Roderick Molenaar, Onno W Steenbeek, and Peter Vlaar. Risk models with jumps and time-varying second moments. 2011.

[26] AM van Hekken. *Check This! Empowering people to plan for retirement. The role of persuasive message strategies and readability*. PhD thesis, [Sl: sn], 2018.

[27] Meredith Whittaker, Kate Crawford, Roel Dobbe, Genevieve Fried, Elizabeth Kaziunas, Varoon Mathur, Sarah Mysers West, Rashida Richardson, Jason Schultz, and Oscar Schwartz. *AI now report 2018*. AI Now Institute at New York University, 2018.