A Neural Network Approach to Efficient Valuation of Large VA Portfolios

by

Seyed Amir Hejazi

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

A Neural Network Approach to Efficient Valuation of Large VA Portfolios

Seyed Amir Hejazi

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2016

Variable annuity (VA) products expose insurance companies to considerable risk because of the guarantees they provide to buyers of these products. Managing and hedging the risks associated with VA products requires intraday valuation of key risk metrics for these products. The complex structure of VA products and computational complexity of their accurate evaluation has compelled insurance companies to adopt Monte Carlo (MC) simulations to value their large portfolios of VA products. Because the MC simulations are computationally demanding, especially for intraday valuations, insurance companies need more efficient valuation techniques.

Existing academic methodologies focus on fair valuation of a single VA contract, exploiting ideas in option theory and regression. In most cases, the computational complexity of these methods surpasses the computational requirements of MC simulations. Recently, a framework based on Kriging has been proposed that can significantly decrease the computational complexity of MC simulation. Kriging methods are an important class of spatial interpolation techniques. In this thesis, we study the performance of prominent traditional spatial interpolation techniques. Our study shows that traditional interpolation techniques require the definition of a distance function that can significantly impact their accuracy. Moreover, none of the traditional spatial interpolation techniques provide all of the key properties of accuracy, efficiency, and granularity. Therefore, in this thesis, we present a neural network approach for the spatial interpolation framework that affords

an efficient way to find an effective distance function. The proposed approach is accurate, efficient, and provides an accurate granular view of the input portfolio. Our numerical experiments illustrate the superiority of the performance of the proposed neural network approach in estimation of the delta value and also the solvency capital requirement for large portfolios of VA products compared to the traditional spatial interpolation schemes and MC simulations.

Dedication

To my wife Narges, my parents Mahnaz and Ali, my sister Asma, and my unborn children who will see this in the future.

Acknowledgements

I want to start by expressing my sincere gratitude to my supervisor Prof. Kenneth R. Jackson who has been a tremendous source of wisdom, guidance, kindness, and encouragement throughout my PhD studies. Being his student and having the unique opportunity of collaborating with him, I have become a better scientist.

I would like to thank the members of my PhD supervisory committee: Prof. Christina Christara, Prof. Sheldon Lin, and Prof. Guojun Gan, for their insightful comments and invaluable advice that enriched and strengthened this thesis. I would also like to thank Prof. Yuying Li for accepting to be my external examiner and Prof. Tom Fairgrieve for agreeing to be a member of my final oral committee.

I am grateful to the Computer Science department at the University of Toronto, the Natural Science and Engineering Research Council of Canada (NSERC), the Government of Ontario, and Greg Wolfond for financially supporting my PhD studies.

Furthermore, I wish to thank my CS family, Venkatesh Medabalimi, George Amvrosiadis, Nosayba El-Sayed, Sahil Suneja, Daniel Fryer, Andy Hwang, Bogdan Simion, Ioan Stefanovici, Larry Zhang, Patricia Thaine, Sean Robertson, Aditya Bhargava, Samir Hamdi, Michael Chiu, and Vida Heidarpour, for their sage advice and support, and for all the fun that we had together. They are the true colors of the Computer Science department.

I want to thank my wonderful friends, Behrooz Abiri, Saeid Rezaei, Kianoosh Hosseini, Amir Hassan Asgari, Aynaz Vatankhah, Milad Eftekhar, Amirali Salehi, Soheil Hassas Yeganeh, Hossein Kaffash Bokharaei, Monia Ghobadi, Mahdi Lotfinezhad, Amin Behnad, Amin Heidari, Hossein Kassiri, Sadegh Jalali, Seyed Hossein Seyedmehdi, Mohammad Shakourifar, and my friends at the UTDisco group, for their love and support. In particular, I am grateful to my dearest friend Afshar Ganjali who has been a tremendous source of support in this hard long journey. He is the brother that I always wished for.

Special thanks go to my parents, Mahnaz Tahmasebi and Seyed Aliakbar Hejazi, my sister Asma Hejazi and her husband Salman Mohazabieh, my mother-in-law Soheila Sartaj, my father-in-law Mohammadreza Balouchestani-asli, and my sister-in-law Negar Balouchestani-asli, for their prayers, their sacrifices, their endless love, and for putting their confidence in me. I am forever indebted to them.

Last but not the least, I want to extend my utmost gratitude to my lovely wife Narges Balouchestani-asli. She is the true accomplishment of my PhD journey. She was the light that I pursued during the most difficult time of this journey when everything else was dark. I am able to successfully finish this long endeavor because of her never ending love and care. Thanks darling for everything.

Contents

List of Tables

List of Figures

Acronyms

AC Available Capital

AV Account Value

CDF Cumulative Distribution Function

CEIOP Committee of European Insurance and Occupational Pensions Supervisors

GD Guaranteed Death Benefit Value

GV Guarantee Value

IDW Inverse Distance Weighting

LHS Latin Hypercube Sampling

LSMC Least Squares Monte Carlo

Mat Maturity

MVA Market Value of Assets

MVL Market Value of Liabilities

MSE Mean Squared Error

MC Monte Carlo

NAG Nestrov's Accelerated Gradient

RBF Radial Basis Function

RDC Randomized Dependence Coefficient

RSS Replicated Stratified Sampling

SCR Solvency Capital Requirement

VA Variable Annuity

WR Withdrawal Rate

Chapter 1

Introduction

Variable annuities are unit-linked products that are wrapped with a life insurance contract. These products allow a policyholder to invest into pre-defined sub-accounts set up by the insurance company. Sub-account funds are invested in bonds, the money market, stocks and other financial products. An insurer offers different types of sub-accounts that are tailored to the appetite of policyholders with different levels of tolerance for risk. The investment can be made via a lump-sum payment or a series of investment purchases. In return, the insurance company offers tax sheltered growth and guarantees that protect the policyholder in a bear market [19].

Upon entering into a contract, the policyholder is given two accounts: the first keeps track of the performance of investments in the sub-accounts while the second keeps track of the amount of guarantee provided by the insurance company. The value of the first account is called the account value and the value of second account is called the benefit base. During a period called the accumulation phase, the policyholder accumulates assets on his investments in sub-accounts and the value of his benefit base appreciates by contractually agreed roll ups, ratchets and resets without taxation. At the end of the accumulation phase, the benefit base is locked in and the insurer guarantees to return at least the benefit base as a lump sum payment or as a stream of payments during a

period called the withdrawal phase.

The most prevalent of the guarantees are the Guaranteed Minimum Death Benefit (GMDB), the Guaranteed Minimum Withdrawal Benefit (GMWB), the Guaranteed Minimum Income Benefit (GMIB), and the Guaranteed Minimum Accumulation Benefit (GMAB). The GMDB guarantees a specified lump sum payment on death regardless of the performance of the underlying account. The most basic guarantee offered now is the return of the benefit base adjusted for any partial withdrawals. The GMWB guarantees the ability to partially withdraw up to a pre-determined percentage (called the withdrawal rate) of the benefit base for a specified number of years. The decision to withdraw is made annually and the maximum amount of withdrawal is a function of the age of the policyholder. The GMIB guarantees a stream of income for life contingent on the survival of the policyholder, and the GMAB guarantees a lump sum payment on maturity of the contract regardless of the performance of the underlying funds. For further details, see [68].

Embedded guarantees are the key selling feature of VA products. These guarantees have allowed insurance companies to sell trillions of dollars worth of these products worldwide, in 2010 alone [43]. As a result, major insurance companies are now managing large portfolios of VA contracts, each with hundreds of thousands of contracts.

Although the embedded guarantees are attractive features to the buyer of VA products, they expose the insurers to substantial risk (e.g., market and behavioral risk). Because of that, major insurance companies have started risk management and hedging [9, 36] programs, especially after the market crash of 2008, to reduce their exposures. An integral part of a risk management program is finding the value of key statistical risk indicators, e.g., the Greeks [42] and the Solvency Capital Requirement (SCR) [4], on daily, monthly and quarterly bases.

Most of the academic research to date has focused on fair valuation of individual VA contracts [2, 8, 13, 17, 18, 22, 24, 27, 34, 41, 46, 51, 52, 70]. Most of the methodologies

developed in these research papers are based on ideas from option pricing theory, and are tailored to the type of VA studied. In addition, almost all of the proposed schemes are computationally expensive and the results they provide for a VA contract cannot be re-used for another VA contract, even of similar type. Each VA contract is unique in terms of its key attributes, i.e., age, gender, account value, guaranteed value, maturity of contract, fund type, etc. Hence, VA portfolios are non-homogeneous pools of VA contracts, and academic methodologies cannot scale well to be used to calculate key risk statistics of large VA portfolios.

Although the nature of the guarantees in the VA products makes them path dependent, in practice, insurance companies relax the assumptions on guarantees and rely heavily on stochastic simulations to value these products and manage their risks. In particular, nested MC simulations are the industry standard methodology in determining key risk metrics [4,61]. A Nested simulation consists of an outer loop that spans the space of key market variables (risk factors) and an inner loop that projects the liability of each VA contract along many simulated risk-neutral paths [29]. As explained in Chapters 2 and 4, MC simulations are computationally demanding, forcing insurance companies to look for ways to reduce the computational load of MC simulations.

1.1 Main Contributions

The unique structure of VA contracts does not allow one to blindly re-use the calculated value of a risk metric for one VA contract for another VA contract. But the calculated value for a sample VA contract can provide partial information (e.g., a value range) for the value of the risk metric for VA contracts that have similar attributes and attribute values as the sample VA contract.

The proposed Kriging methods in [30, 32] incorporate the partial information of a sample portfolio of VA contracts for which the values of key risk metrics are known and

approximate the values of the key risk metrics for other VA contracts. As we explain in Chapter 2, the proposed Kriging methods can be categorized under the general framework of spatial interpolation. Spatial interpolation techniques improve the efficiency of evaluation for large VA portfolios by reducing the number of VA contracts for which the MC simulations must be performed.

In this thesis, we provide an extensive study of the spatial interpolation framework when employed with existing interpolation schemes to approximate the Greeks for large portfolios of VAs. Our experimental results, in Chapter 2, show that none of the existing interpolation schemes we considered achieves all of the desired requirements of efficiency, accuracy and granularity (per policy view of the portfolio). We explain in Chapters 2 and 3 that a major drawback of existing interpolation techniques is their reliance on a distance function. Defining a good distance function is not straightforward and requires input from a subject matter expert. The amount of time invested by a subject matter expert to define the choice of distance function may be significant and hence diminish the efficiency of the framework.

In Chapter 3, we use ideas in machine learning theory to propose the Nadaraya-Watson estimators [54,72] as the class of spatial interpolation techniques that can provide all of efficiency, accuracy and granularity. Finding an appropriate choice of distance function, in this class of estimators, can be formulated as an optimization problem on the choice of bandwidth parameters and the function that is used in these estimators to do the Parzen density estimations. As we discuss in Chapter 3, the original definition of Nadaraya-Watson estimators uses a universal set of bandwidth parameters. However, in our application of interest, it is better to allow the bandwidth parameters to be dependent on the location of Parzen density estimators. Hence, we propose a formulation of the Nadaraya-Watson estimators that uses location dependent bandwidth parameters. This formulation allows us to use a single layer neural network to implement our proposed extension of the Nadaraya-Watson estimator.

The proposed neural network learns an appropriate distance function through a calibration (training) process that finds an appropriate choice of the bandwidth parameters. To significantly reduce manual input by a subject matter expert, we propose a design of the calibration process that can detect when to stop the calibration process and how to deal with the issue of overfitting. Our experiments in Chapter 3 demonstrate that the framework provides approximate values of the key risk metrics at a micro (per policy) level and at a macro (portfolio) level in a fast, accurate way. The proposed neural network, in comparison to the existing spatial interpolation techniques has better accuracy and comparable efficiency. The neural network, for a comparable accuracy, is faster than a straightforward MC simulation in estimation of the delta value of a large portfolio of $100,000$ VA contracts, by a factor greater than 15.

As we discuss in Chapter 3, the training stage of the proposed neural network approach is time consuming, which may not be ideal if the neural network is used for intraday valuations. However, a small change in market conditions usually does not affect the key risk metrics of VA policies significantly. Therefore, a small change in the parameters of the trained neural network should be sufficient to preserve the accuracy of the estimations under the new market conditions. We build on this idea and provide a solution in Chapter 4 that decreases the training time of the network significantly.

In Chapter 4, we use the neural network to estimate the one-year loss probability distribution for a large portfolio of VA products, which is an important step in computing the SCR for the portfolio. To estimate the one-year loss probability distribution, we have to compute liabilities of the input portfolio under many market conditions. A MC approach, as suggested in [4], to compute the one-year loss probability distribution, even with a parallel processing implementation, admits significant computational complexity. We show in Chapter 4, how the proposed neural network approach can provide an efficient and accurate alternative. Our numerical experiments show that a sequential implementation of the proposed neural network approach can compute the SCR for a portfolio of

100,000 VA policies at a speed that is 6 times faster than a parallel implementation of MC simulations.

We discuss in Chapter 3 that a bad choice of representative contracts can cause a severe deterioration in the accuracy of our neural network framework. Furthermore, our numerical experiments show that a bad choice of the representative portfolio or the training portfolio can lead to a poor training of the network parameters and can increase the training time of the neural network.

To ameliorate the above-mentioned problem, in Chapter 5, we propose a novel sampling method that uses the distribution of the input portfolio in the space in which it is defined to generate VAs for the representative portfolio and the training portfolio. Our experiments in Chapter 5 show that the proposed sampling method, when used in the neural network framework of Chapter 3, can accurately and efficiently estimate the delta value of synthetic portfolios of VAs that are uniformly and/or non-uniformly distributed in the space.

1.2 Thesis Outline

In Chapter 2, we review the existing methodologies used to value key risk metrics of a large portfolio of VAs. In particular, we discuss the pitfalls of these approaches that may prevent them from efficiently computing accurate estimates of key risk metrics. We describe the spatial interpolation framework that can significantly decrease the computational complexity of MC simulations by reducing the number of policies that undergo MC simulations. We study the performance of this framework when employed with existing interpolation schemes. Our numerical results show that the existing interpolation schemes can provide only a subset of performance metrics (accuracy, efficiency, and granularity). We also discuss the importance of the choice of sampling methods and the distance function on the performance of the spatial interpolation scheme.

In Chapter 3, we propose a neural network approach to estimate the key risk metrics. The proposed neural network learns the choice of a good distance function and hence eliminates the need for a manual input of a distance function by a subject matter expert. Our numerical results corroborate that the neural network provides accurate estimates of the key risk metrics at both the policy level and the portfolio level in an efficient manner.

In Chapter 4, we use the proposed neural network to compute the SCR for a large portfolio of VAs. A bottleneck in the performance of the proposed neural network approach is the training time. We demonstrate, in Chapter 4, that small updates to the parameters of a trained network to find optimal parameter values for a new network can significantly reduce the training time.

In Chapter 5, we propose a sampling method that generates portfolios of sample VA policies that have distributions in space that are similar to the distribution of the input portfolio of VAs. The method allows for better training of the neural network and enhances the accuracy of the neural network framework for non-uniformly distributed input portfolios.

Finally, Chapter 6 concludes the thesis and discusses possible future directions.

Chapter 2

Application of Spatial Interpolation in Estimation of Greeks

In this chapter[1], we focus on the efficient approximation of the Greeks for large portfolios of VA products. In particular, we provide an extensive study of a framework based on metamodeling that can approximate the Greeks for large portfolios of VAs in a fast and accurate way.

2.1 Portfolio Valuation Techniques

If one thinks of VAs as exotic market instruments [42], the traditional replicating portfolio approach can be used to find the value of a portfolio of VA products. The main idea behind this approach is to approximate the cash flow of liabilities for a portfolio of VA contracts using well-formulated market instruments such as vanilla derivatives. The problem is often formulated as a convex optimization problem where the objective is to minimize the difference between the cash flows of the input portfolio and the replicating portfolio. Depending on the norm associated with the problem, linear programming [26]

[1]The material of this chapter is based on our paper [39].

or quadratic programming [25,57] is used in the literature to find the replicating portfolio. The replicating portfolio, in our application of interest, doesn't provide us with an efficient alternative to MC simulations, as one still needs to find the cash flow of the input portfolio for each year up to maturity.

Least Squares Monte Carlo (LSMC) regresses the liability of the input portfolio against some basis functions representing key economic factors [14, 47]. LSMC has been proposed in the literature to reduce the number of inner loop scenarios in nested MC simulations [15]. Depending on the type of embedded guarantees, size of investment and characteristics of the policyholder, VA contracts have many numeric attributes, each covering a broad range. Therefore, an accurate regression using LSMC requires incorporating many sample points in the space of key risk factors, and hence is computationally demanding.

In [11], another regression approach that is similar to LSMC is proposed to reduce the computational complexity associated with the calculation of risk measures associated with the loss distribution of financial portfolios. Although this method is successful in increasing the convergence rate of MC simulations and in reducing the number of inner loop scenarios to as low as one, it is applicable to only a very limited number of risk metrics that can be defined as an expectation of a function of the loss. In particular, the method does not apply to important risk measures such as Greeks, VaR and Solvency Capital Requirement. Furthermore, achieving the desired level of accuracy with only one inner loop scenario for each outer loop scenario still requires many outer loop scenarios.

Recently, Replicated Stratified Sampling (RSS) [71] and Kriging based techniques [30, 32] have been proposed to reduce the number of VA contracts that must be included in the MC simulations. Both of these methods, use the Greeks for samples of the input portfolio to estimate the Greeks of the full input portfolio. RSS requires several iterations of sample generation and evaluation to converge to a final result. This makes it more computationally demanding than the Kriging based techniques of [30, 32] that require

MC simulations results for only one sample.

The Kriging based methods of [30, 32], first, select a small set of representative VA policies, using various data clustering [33] or sampling methods, and price only these representative policies via MC simulations. The representative contracts and their Greeks are then fed as training samples to a machine learning algorithm [5] called Kriging [23], which then estimates the Greeks of the whole portfolio. In the rest of the chapter, we provide a study of the more general framework of spatial interpolation, including Kriging methods, and provide more insights into why spatial interpolation can be much more efficient and accurate than other approaches in the literature.

In this thesis, we use the term interpolation in the general context of estimating the values at unknown locations using known data at a finite number of interpolation points. In this context, an interpolation method for which the predicted values are exactly equal to the known values at all the interpolation points is called an *exact* interpolator. Interpolation methods that do no satisfy this constraint are called *inexact* interpolation methods [12].

2.2 Spatial Interpolation Framework

The proposed methods in [30–32] can be categorized under the general framework of spatial interpolation. Spatial interpolation is the procedure of estimating the value of data at unknown locations in space given the observations at sampled locations [12]. As the definition suggests, spatial interpolation requires a sampling method to collect information about the surface of interest and an interpolation method that uses the collected information to estimate the value of the surface at unknown locations. As discussed in [30–32], the choice of sampling method and interpolation method can noticeably impact the quality of the interpolation. In this chapter, we choose to focus on the latter, and leave a discussion of the choice of an appropriate sampling method to Chapter 5.

In the functional data analysis literature, there exist two main classes of interpolation methods [12]:

- **Deterministic Interpolation:** Creates surfaces from measured points on the basis of either similarity or degree of smoothness.
- **Stochastic Interpolation:** Utilizes statistical properties of measured points, such as auto-correlation amongst measured points, to create the surface.

In what follows, we study three (one stochastic, and two deterministic) of the most prominent of these interpolation techniques— Kriging, Inverse Distance Weighting (IDW) and Radial Basis Function (RBF)— in the context of our problem of interest. In particular, we study these interpolation techniques to estimate the delta value for a large portfolio of VA products. Although our study focuses on estimation of the delta value, the framework is general and can be applied to estimate other Greeks as well. We compare the performance of these methods in terms of computational complexity and accuracy at the micro (contract) level and at the macro (portfolio) level.

Although [30,32] provide some insights into the performance of the Kriging interpolation methods, we provide further insights into the efficiency and accuracy of Kriging based methods in comparison to other interpolation techniques. Moreover, we shed some light on how Kriging achieves its documented performance and discuss some issues regarding the choice of variogram model and distance function.

2.2.1 Sampling Method

In this chapter, we focus on studying synthetic portfolios that are generated uniformly at random in the space of selected variable annuities. In [31], the Latin Hypercube Sampling (LHS) method [50] is used to select representative contracts. LHS provides a uniform coverage of the space including the boundary VA contracts. The results of [31] indicate that LHS increases the accuracy of the estimation compared to other sampling

methods. In order to preserve the properties of LHS, in this chapter, we select our representative contracts by dividing the range of each numeric attribute of a VA contract into almost equal length subintervals, selecting the end points of resulting subintervals and producing synthetic contracts from all combinations of these end points and choices of categorical attributes.

2.2.2 Kriging

Kriging is a stochastic interpolator that gives the best linear unbiased estimation of interpolated values assuming a Gaussian process model with prior covariance [45, 49]. Various Kriging methods (i.e., Simple Kriging, Ordinary Kriging, Universal Kriging, etc.) have been developed based on assumptions about the model. In our experiments, we didn't find any significant advantages in choosing a particular Kriging method. Therefore, for the sake of simplicity of analysis, and based on the results of [32], we focus on ordinary Kriging in this thesis.

Assume $Z(x)$ represents the delta value of a VA contract represented in space by the point x. Let $Z(x_1), Z(x_2), \ldots, Z(x_n)$ represent the observed delta values at locations $x_1, x_2, \ldots, x_n$. Ordinary Kriging tries to find the best, in the Mean Squared Error (MSE) sense, unbiased linear estimator $\hat{Z}(x) = \sum_{i=1}^{n} \omega_i Z(x_i)$ of $Z(x)$ by solving the following system of linear equations to find the w_is.

$$\begin{bmatrix} \gamma(D(x_1,x_1)) & \gamma(D(x_1,x_2)) & \ldots & \gamma(D(x_1,x_n)) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma(D(x_n,x_1)) & \gamma(D(x_n,x_2)) & \ldots & \gamma(D(x_n,x_n)) & 1 \\ 1 & 1 & \ldots & 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \\ \lambda \end{bmatrix} = \begin{bmatrix} \gamma(D(x_1,x)) \\ \ldots \\ \gamma(D(x_n,x)) \\ 1 \end{bmatrix} \tag{2.1}$$

where λ is the Lagrange multiplier [7], $\gamma(\cdot)$ is the semi-variogram function, to be discussed shortly, and $D(\cdot,\cdot)$ is a distance function that measures the distance between two points in the space of VA contracts. The last row enforces the following constraint to allow an

unbiased estimation of $Z(x)$.

$$\sum_{i=1}^{n} w_i = 1 \tag{2.2}$$

Note that w_i is a function of $x, x_1, \cdots, x_n$, but, to simplify the notation, we write it as w_i throughout this chapter.

In this formulation of the Kriging problem, the system of linear equations (2.1) should be solved once for each VA policy (point in space). Solving a system of linear equations, with standard methods, takes $\Theta(n^3)$[2] time. Hence, estimating the delta value for a portfolio of N VA contracts by summing the estimated delta value of each contract requires $\Theta(N \times n^3)$ time which is computationally expensive. Because of this, Kriging methods are inefficient in finding a granular view of the portfolio. However, if we are only interested in the Greeks of the portfolio, and not the Greeks of each individual policy, we can follow the approach of [30–32] and use the linearity of the systems of linear equations to sum them across the portfolio in $\Theta(n \times N)$ and to solve only the resulting system of linear equations in time proportional to n^3. Hence estimating the delta of a portfolio requires $\Theta(n^3 + n \times N)$ time. To sum the systems of linear equations, we sum the corresponding weights and Lagrange multipliers on the left side of the equations and sum the corresponding terms, i.e., $\gamma(D(x_i, x)), i = 1, 2, \ldots, n$, and constants, on the right side of the equations.

Variogram

Kriging assumes the Gaussian process $Z(x)$ is second order stationary, i.e., the covariance of the Gaussian process in two locations is a function of the distance between the two locations. Assuming a zero mean, the Gaussian process covariance function can be defined in terms of a variogram function $2\gamma(h)$

[2] $f(x) = \Theta(g(x))$ means that there exists positive numbers c_1, c_2, and M such that $\forall x > M : c_1 g(x) \leq f(x) \leq c_2 g(x)$.

$$\begin{aligned} Cov_Z(x+h,x) =& E\big[Z(x+h)Z(x)\big] \\ =& \frac{1}{2}E\Big[Z^2(x+h)+Z^2(x)-\Big(Z(x+h)-Z(x)\Big)^2\Big] \\ =& Var(Z)-\frac{1}{2}(2\gamma(h)) \end{aligned} \tag{2.3}$$

In practice, for a set of sample points $x_i, 1 \leq i \leq n$, the variogram can be estimated as

$$2\hat{\gamma}(h) = \frac{1}{N(h)}\sum_{i=1}^{N(h)}\big(Z(x_i+h)-Z(x_i)\big)^2 \tag{2.4}$$

where $N(h)$ is the number of pairs in the sample separated by a distance h from each other. The function $2\hat{\gamma}(h)$ is often called the *empirical variogram*.

Because of the noise in measurements, the estimated empirical variogram may not represent a valid variogram function. Since methods like Kriging require valid variograms at every distance h, empirical variograms are often approximated by model functions ensuring the validity of the variogram [20]. Variogram models are usually described in terms of three important variables:

- **Nugget (n):** The height of the discontinuity jump at the origin.
- **Sill (s):** The Limit of the variogram as the lag distance h approaches infinity.
- **Range (r):** The distance at which the difference of the variogram from the sill becomes negligible.

Figure 2.1 shows an example of an empirical variogram and the model variogram. In our study, we choose to focus on the following three prominent variogram models [20, 23]:

- Exponential Variogram

$$\gamma(h) = (s-n)\Big(1-\exp\big(-\frac{h}{(ra)}\big)\Big)+n\mathbf{1}_{(0,\infty)}(h)$$

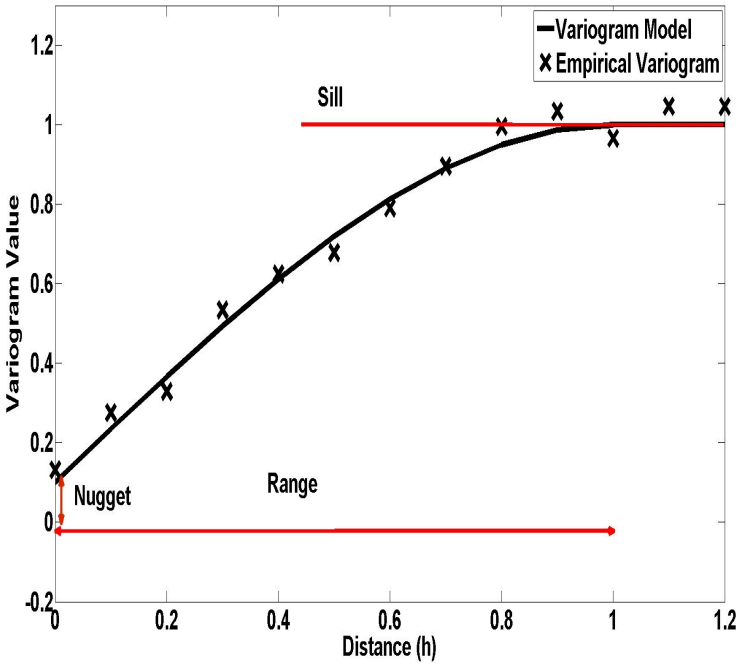


Figure 2.1: Example of a variogram.

- Spherical Variogram

$$\gamma(h) = (s-n)\Big((\frac{3h}{2r} - \frac{h^3}{2r^3})\mathbf{1}_{(0,r)}(h) + \mathbf{1}_{[r,\infty)}(h)\Big) + n\mathbf{1}_{(0,\infty)}(h)$$

- Gaussian Variogram

$$\gamma(h) = (s-n)\Big(1 - \exp\big(-\frac{h^2}{r^2 a}\big)\Big) + n\mathbf{1}_{(0,\infty)}(h)$$

In Exponential and Gaussian variogram models, a is a free parameter that is chosen so that the variogram better fits the data.

2.2.3 Inverse Distance Weighting

Inverse Distance Weighting (IDW) is a deterministic method that estimates the value at an unknown position x as a weighted average of values at known positions $x_1, \ldots, x_n$. Assuming the delta values $Z(x_1), Z(x_2), \ldots, Z(x_n)$ of representative VAs $x_1, x_2, \ldots, x_n$, we can estimate the delta value $Z(x)$ of a VA at a point x as

$$\hat{Z}(x) = \begin{cases} \frac{\sum_{i=1}^{n} w_i(x) Z(x_i)}{\sum_{i=1}^{n} w_i(x)} & \forall i : D(x, x_i) \neq 0 \\ Z(x_i) & \exists i : D(x, x_i) = 0 \end{cases} \tag{2.5}$$

where $w_i(x) = D(x, x_i)^{-p}$, and $D(\cdot, \cdot)$ is a distance function [63]. The parameter p in $w_i(x)$ is a positive real number called the "power parameter". The choice of power parameter depends on the distribution of samples and the maximum distance over which an individual sample is allowed to influence the surrounding points. Greater values of p assign greater influence to values closest to the interpolating point. The choice of the power parameter also influences the smoothness of the interpolator by changing the influence radius of sample points.

In comparison to Kriging, IDW requires only $\Theta(n)$ operations to estimate the delta value of each new VA contract using the delta values of n representative contracts. Assuming a portfolio of N VA contracts, we can estimate the delta value of the portfolio by summing the estimated delta value of contracts in time proportional to $n \times N$. Hence, we expect IDW to be faster than Kriging to estimate the delta value of the portfolio. The difference in speed is more apparent if we want a more granular view of the portfolio. In other words, if we are interested in the estimated delta value of each VA contract in the portfolio, Kriging is much slower than IDW. We provide further insights into this matter in Section 2.3.

2.2.4 Radial Basis Functions

In the RBF method, we approximate the delta value of a VA contract x as a weighted linear combination of radial functions centered at representative contracts $x_1, x_2, \ldots, x_n$:

$$\hat{Z}(x) = \sum_{i=1}^{n} w_i \Phi(||x - x_i||) \tag{2.6}$$

where $|| \cdot ||$ is a norm, usually chosen to be Euclidean distance.

In RBF interpolation, the weights are chosen so that RBF is exact at the $x_i, 1 \leq i \leq$

n, points. In other words, given the values $Z(x_1), \ldots, Z(x_n)$ at points $x_1, \ldots, x_n$, the following linear set of equations is solved for w_i:

$$\begin{bmatrix} \Phi(||x_1 - x_1||) & \ldots & \Phi(||x_1 - x_n||) \\ \vdots & \ddots & \vdots \\ \Phi(||x_n - x_1||) & \ldots & \Phi(||x_n - x_n||) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} Z(x_1) \\ \vdots \\ Z(x_n) \end{bmatrix} \tag{2.7}$$

In our research, we chose the following commonly used radial basis functions

- Gaussian

$$\Phi(x) = \exp(-\epsilon x^2) \tag{2.8}$$

- Multi-Quadratic

$$\Phi(x) = \sqrt{1 + (\epsilon x)^2} \tag{2.9}$$

These two functions represent two classes of radial basis functions: 1) the class in which the value of the radial function increases with the distance from its center, 2) the class in which the value of radial function decreases with the distance form its center. Although the latter class of RBF functions, which is represented by the Gaussian radial function in our study, seems more suitable for our application of interest, for the sake of completeness, we chose to experiment with the former class as well in our study. In both of the above-mentioned functions, ϵ is a free parameter that defines the significance of known points on the value of their neighbor points.

Similar to IDW, RBF interpolation has a running time that is proportional to n for the delta value estimation of each VA contract, and a running time of $\Theta(n \times N)$ to estimate the delta value of a portfolio of N VA contracts. But in addition we need extra $\Theta(n^3)$ time to solve (2.7). Hence, in total, the computational complexity of RBF interpolation to estimate the delta value of a portfolio is $\Theta(n \times N + n^3)$. Similar to IDW, the RBF interpolation can provide us more granularity in less time than the Kriging method.

Attribute	Value
Guarantee Type	{GMDB, GMDB + GMWB}
Gender	{Male, Female}
Age	$\{20, 21, \ldots, 60\}$
Account Value	$[1e4, 5e5]$
Guarantee Value	$[0.5e4, 6e5]$
Withdrawal Rate	$\{0.04, 0.05, 0.06, 0.07, 0.08\}$
Maturity	$\{10, 11, \ldots, 25\}$

Table 2.1: GMDB and GMWB attributes and their respective ranges of values.

2.3 Numerical Experiments

In this section, we present numerical results on the performance of each interpolation method in the context of the proposed framework. In all of our experiments, our goal is to estimate the delta value of a synthetic portfolio of $100,000$ VA contracts which are chosen uniformly at random from the space defined by attributes listed in Table 2.1. The range of attributes are similar to the ones reported in [30,32] which allows us to compare fairly our results with the reported findings in [30,32]. However, for the sake of generality, we allow VA contracts, with guarantee values that are not equal to the account value. Moreover, for VA contracts with a GMWB rider, we set the guaranteed death benefit value to be equal to the guaranteed withdrawal benefit.

In our experiments, we use the framework of [3] to value each VA contract, and assume the output of a MC simulation with $10,000$ inner loop scenarios as the actual value of the contract. The reason behind our choice is that, when fewer inner loop scenarios are used, e.g., 1000 as used in [30], we observed a noticeable difference, as big as 5%, between the computed portfolio delta value from successive runs. However, when we used $10,000$ inner loop scenarios, the absolute maximum difference in the estimated portfolio delta

values of any two runs (in about 20 runs) of the MC simulations was at most 0.93% of the mean value of the portfolio delta estimations. The standard deviation of these estimations was 0.25% of the mean value of the portfolio delta estimations. Inner loop scenarios are generated assuming a simple log-normal distribution model [42] with a risk free rate of return of $\mu = 3\%$, and volatility of $\sigma = 20\%$. Our mortality rates follow the 1996 IAM mortality tables provided by the Society of Actuaries.

In the training (calibration) stage of our proposed framework, we use MC simulations with $10,000$ inner loop scenarios to find the delta value of our representative contracts.

2.3.1 Performance

In this set of experiments, our objective is to provide a fair comparison of accuracy and computational efficiency of each proposed estimation method when the k-prototype distance function of [30] is used. Since we allow the guaranteed value of VAs in the synthetic portfolio to be different than their account value, we have modified the distance function as follows:

$$D(\mathbf{x}, \mathbf{y}, \gamma) = \sqrt{\sum_{h \in N} (\frac{x_h - y_h}{\max_h - \min_h})^2 + \gamma \sum_{h \in C} \delta(x_h, y_h)} \tag{2.10}$$

where $N = \{$AV, GD, GW, maturity, age, withdrawal rate$\}$ is the set of numerical values and $C = \{$gender, rider$\}$ is the set of categorical values.

Similar to [30, 32], we choose $\gamma = 1$. Moreover, we form the set of representative contracts, via the sampling method of Section 2.2.1, from all combinations of end points presented in Table 2.2. Because of the constraints on the guaranteed values, some of the entries are duplicate, which we remove to obtain a sample of size 1800.

In order to be thorough in our experiments and comprehensive in our analysis, we present the results for all variants of the Kriging, IDW, and RBF methods. For Kriging, we choose to experiment with all three major variogram models, i.e., spherical, exponential, and Gaussian. For IDW, we choose to experiment with different choices of the

Attribute	Value
Guarantee Type	{GMDB, GMDB + GMWB}
Gender	{Male, Female}
Age	$\{20, 30, 40, 50, 60\}$
Account Value	$\{1e4, 1.25e5, 2.5e5, 3.75e5, 5e5\}$
Guarantee Value	$\{0.5e4, 3e5, 6e5\}$
Withdrawal Rate	$\{0.04, 0.08\}$
Maturity	$\{10, 15, 20, 25\}$

Table 2.2: Attribute values from which representative contracts are generated for experiments.

power parameter to see the effect of this free parameter on the accuracy of results. For RBF, we study two of the most popular radial functions, Gaussian and multi-quadratic, and for each type of radial function, we experimented with two choices for the free parameter ϵ. Although in this Chapter and Chapter 3, we report only the results of our experiments with RBF interpolation, we did experiment with RBF regression. However, the performance of RBF regression was close to that of RBF interpolation and, in some cases, even worse. To improve the performance of RBF regression, we need to study the choice of regularization parameters and the training data, which significantly adds to the complexity of the approach. Furthermore, RBF regression, like RBF interpolation, suffers from the problem that it requires a proper choice of distance function– for a more detailed description and analysis of this problem, refer to Section 2.3.3. Therefore for tractability of our analysis, in this thesis, we only choose to report the results of our experiments with RBF interpolation.

In Table 2.3, the relative error in estimation of the delta value of the portfolio is presented. The relative error for method m is calculated as follows.

Method	Relative Error (%)
Kriging (Spherical)	-0.03
Kriging (Exponential)	-1.61
Kriging (Gaussian)	< -500
IDW (p = 1)	9.11
IDW (p = 10)	13.12
IDW (p = 100)	11.99
RBF (Gaussian, $\epsilon = 1$)	-1.79
RBF (Gaussian, $\epsilon = 10$)	37.89
RBF(Multi-Quad, $\epsilon = 1$)	-71.62
RBF(Multi-Quad, $\epsilon = 10$)	-10.86

Table 2.3: Relative error in estimation of the delta value via each method.

$$\text{Err}_m = \frac{\Delta_m - \Delta_{MC}}{|\Delta_{MC}|} \tag{2.11}$$

where Δ_{MC} is the estimated delta value of the portfolio computed by MC simulations and Δ_m is the estimate delta value of the portfolio computed by method m. While two of the Kriging methods provide accurate estimates, the accuracy of IDW, and multi-quadratic RBF methods is moderate. One interesting observation is that the choice of variogram model has substantial impact on the accuracy of the Kriging method and it confirms the result of [30] that the spherical method provides the best accuracy. Another interesting observation is the effect of the free parameters p and ϵ on the accuracy of the IDW and RBF methods. The results suggest that the effective use of either method requires a careful tuning of these free parameters.

Table 2.4 presents the running time of each algorithm for two scenarios: 1) estimating the delta value of the portfolio only 2) estimating the delta value of each VA policy in the

portfolio and summing them to get the delta value of the portfolio. While the former does not provide a granular view of the portfolio, the latter gives a more refined estimation process and allows for deeper analysis and insights. Note that the times in Table 2.4 represent only the time that it took to estimate the values once we knew the delta values of the representative contracts. To get the total simulation time, add 187 seconds to these times, which is the time that it took to estimate the delta value of representative contracts via MC simulations. The results show the superiority of the proposed spatial interpolation framework over MC simulation (speed up $> 15\times$) except when Kriging is used for per policy estimation of delta. Because the IDW and RBF methods by definition require the estimation of the delta of each policy and sum the estimations to get the delta value of the portfolio, we can see that simulation times for these methods are approximately equal in the two presented scenarios. Moreover, these methods are more efficient than the Kriging method, which confirms our analysis in Section 2.2.

2.3.2 Accuracy

The accuracy results of Table 2.3 may misleadingly suggest that the Kriging method with the Spherical variogram model is always capable of providing very accurate interpolations. In the experiments of this section, we provide results on the accuracy of different methods that contradict this hypothesis.

For our experiments in this section, we replicated the experiments of Section 2.3.1 with sets of representative contracts that are produced from the set of representative contracts in Section 2.3.1 by removing 100, 200, 400, 600 and 800 VA contracts at random. Table 2.5 presents the mean and the standard deviation of the relative error, in estimation of the delta value of the VA portfolio, for each method in these experiments. The results of Table 2.5 show high variance values for the accuracy of the Kriging methods, which contradicts our hypothesis. Another interesting observation is that the IDW methods and the RBF methods with a Gaussian kernel, in comparison to the Kriging methods,

Method	Portfolio	Per Policy
MC	10617	10617
Kriging (Spherical)	312	> 320000
Kriging (Exponential)	333	> 320000
Kriging (Gaussian)	383	> 320000
IDW (P = 1)	285	286
IDW (P = 10)	288	287
IDW (P = 100)	287	301
RBF (Gaussian, $\epsilon = 1$)	295	306
RBF (Gaussian, $\epsilon = 10$)	294	315
RBF(Multi-Quad, $\epsilon = 1$)	289	289
RBF(Multi-Quad, $\epsilon = 10$)	297	292

Table 2.4: Simulation time for each method to estimate the delta value. All times are in seconds.

have a lower variance value for the relative error.

2.3.3 Distance Function

A key element in the definition of each estimation method is the choice of a distance function. While the RBF method requires the choice of a proper distance function, Kriging and IDW can work with any choice of distance function. A proper distance function satisfies non-negativity, identity of indiscernibles, symmetry and the triangle inequality [66]. We call any function that has the non-negativity property and a subset of other aforementioned properties a distance function. In this set of experiments, we investigate the importance of the choice of distance function on the accuracy of estimation for each interpolation method.

Method	Mean (%)	STD (%)
Kriging (Spherical)	0.47	1.76
Kriging (Exponential)	-0.58	2.19
Kriging (Gaussian)	1109.02	3289.81
IDW (p = 1)	9.14	1.75
IDW (p = 10)	13.14	0.42
IDW (p = 100)	12.06	0.23
RBF (Gaussian, $\epsilon = 1$)	-1.78	0.48
RBF (Gaussian, $\epsilon = 10$)	38.87	1.42
RBF(Multi-Quad, $\epsilon = 1$)	-58.65	16.84
RBF(Multi-Quad, $\epsilon = 10$)	-9.15	3.56

Table 2.5: Mean and standard deviation of the relative error in estimation of the delta value via each method.

To achieve this goal, we conduct two sets of experiments. In the first set of experiments, we study the effect of the γ variable in (2.10) by reducing the value of γ from 1 to 0.05. γ determines the relative importance of the categorical attributes compared to the numerical attributes, which has not been studied previously. In the second set of experiments, we use the following distance function in our methods with $\gamma = 1$.

$$
\begin{aligned}
D(\mathbf{x}, \mathbf{y}, \gamma) &= \sqrt{f(x_{\text{age}}, y_{\text{age}}) g_{\text{age}}(\mathbf{x}, \mathbf{y}) + \sum_{h \in N} g_{\text{h}}(\mathbf{x}, \mathbf{y}) + \gamma \sum_{h \in C} \delta(x_h, y_h)} \\
f(x_{\text{age}}, y_{\text{age}}) &= \exp\big(\frac{x_{\text{age}} + y_{\text{age}}}{2} - M\big) \\
g_h(\mathbf{x}, \mathbf{y}) &= (\exp(-r_x) x_h - \exp(-r_y) y_h)^2
\end{aligned}
\tag{2.12}
$$

where $C = \{\text{gender, rider}\}$, $N = \{\text{maturity, withdrawal rate}\}$, $r = \frac{AV}{GD}$ and M is the maximum age in the portfolio.

If we view the embedded guarantees in the VA contracts as options that a policyholder can choose to exercise, the ratio r represents the moneyness of that option. If $r \gg 1$, then the account value is enough to cover the amount of guaranteed value. However, if $r \ll 1$, the account value is insufficient to cover the guaranteed value and the insurer has a potential liability. Hence, in estimating the delta value for a VA contract with $r \gg 1$, the delta value is close to zero and the choice of representative contract(s) should not affect the outcome of the estimation as long as the selected representative contract(s) have $r \gg 1$. The choice of function $g(\cdot, \cdot)$ in (2.12) captures the aforementioned idea. In addition, as the age of the policyholder increases, their mortality rate also increases (consult the data of 1996 IAM mortality table). Hence, the liability and delta value of similar contracts which differ only in the age of the policyholder increases with age. Because of this, more emphasis should be placed on estimating the delta value for senior policyholders, which is the motivation behind the introduction of the function $f(\cdot, \cdot)$ in (2.12).

Table 2.6 presents the accuracy of our estimation by each method in both experiments. In experiment one, the choice of $\gamma = 0.05$ has improved the accuracy of most interpolation schemes compared to the results in Table 2.5. Kriging interpolation with a Spherical variogram, the IDW method with $p = 10$, and the RBF method with Gaussian kernel and $\epsilon = 1$ are the only schemes for which the accuracy deteriorated. In experiment two, the Kriging and RBF methods encounter singularities with (2.12); however, the choice of (2.12) has improved the accuracy of the IDW methods. In general, it seems that the choice of distance function and free parameters plays a key role in the accuracy of the interpolation schemes.

2.3.4 Variogram

As mentioned in Section 2.2.2, Kriging methods work with variogram models. The choice of variogram model is dictated by its closeness to the empirical variogram. In the ex-

Method	Relative Error (%)	
	Experiment 1	Experiment2
Kriging (Spherical)	1.94	$*$
Kriging (Exponential)	-0.37	$*$
Kriging (Gaussian)	< -500	$*$
IDW (p = 1)	8.97	-4.87
IDW (p = 10)	13.21	3.90
IDW (p = 100)	11.99	2.32
RBF (Gaussian, $\epsilon = 1$)	-2.56	$*$
RBF (Gaussian, $\epsilon = 10$)	37.89	$*$
RBF(Multi-Quad, $\epsilon = 1$)	-35.74	$*$
RBF(Multi-Quad, $\epsilon = 10$)	-6.88	$*$

Table 2.6: Relative error in the estimation of the delta value by each method. In experiment 1, (2.10) is used with $\gamma = 0.05$, and, in experiment 2, (2.12) is used with $\gamma = 1$. A "$*$" indicates that the method cannot work with this choice of distance function because it causes singularities in the computations.

periments in the previous section, we showed that we can obtain better results using a spherical variogram model; however, we haven't provided any analysis supporting why this variogram is a better choice. In this section, we address this subject. In particular, we conduct experiments to explore whether we can increase the accuracy of the Kriging method by choosing a variogram function that can better approximate the empirical variogram.

To compute the empirical variogram, we partition the x-axis into 20 intervals of equal length $\frac{h_{\max}}{20}$ where $h_{\max}$ is the maximum distance between two VA policies using the distance function (2.10) and with $\gamma = 1$. In each interval, to approximate (2.4), we use the average of the squared difference of the delta value of all pairs of VA policies that have

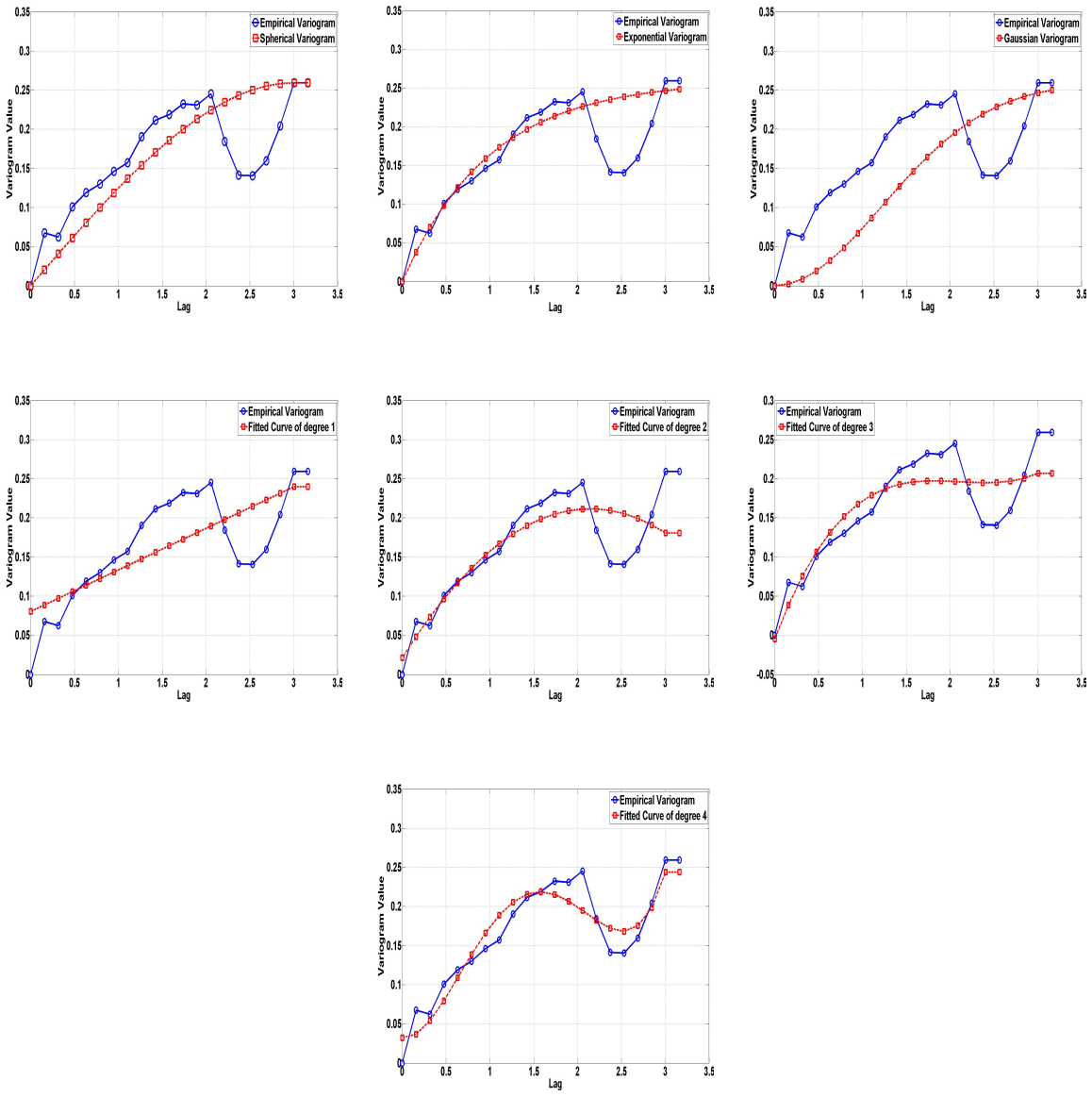


Figure 2.2: Comparing the variogram models with the empirical variogram.

a distance that falls into that interval as the representative for the empirical variogram for that interval. We call the piece-wise linear function that is formed by connecting the representative values for each interval the empirical variogram.

To approximate the empirical variogram, we use polynomials of degree 1, 2, 3 and 4. The polynomials are best MSE approximations of the empirical variogram in the interval

Method	Relative Error (%)
Spherical	-0.03
Exponential	-1.61
Gaussian	< -500
Deg 1	-6.00
Deg 2	-32.17
Deg 3	-2.78
Deg 4	< -500

Table 2.7: Relative error in estimation of the delta value via Kriging with different variogram models.

between zero and the range. At any value greater than the range, the estimated value is assumed to be the value of the polynomial at the range (Figure 2.2). This is necessary in order to have a proper variogram model [23] without producing jumps in the variogram.

The accuracy of Kriging using each approximation of the empirical variogram is presented in Table 2.7. An interesting, yet counter intuitive, observation is that the accuracy of Kriging is worst for the quartic MSE approximation variogram model, which is the approximate variogram model that best fits the empirical variogram. Even comparing the graph of the exponential and spherical variogram models with the empirical variogram, the exponential variogram model seems to fit the empirical variogram model better than the spherical variogram model, but the accuracy of Kriging with the exponential variogram model is worse than the accuracy of Kriging with the spherical variogram model.

Because of these counter intuitive results, we took a closer look at the data from which the empirical variogram was generated. Figure 2.3 shows a graph of the squared differences of delta values of a pair of VA contracts versus their distance from each other. Surprisingly, the point values do not look similar to their average, i.e., the empirical variogram. We expected to see a graph similar to Figure 2.1 where the point values

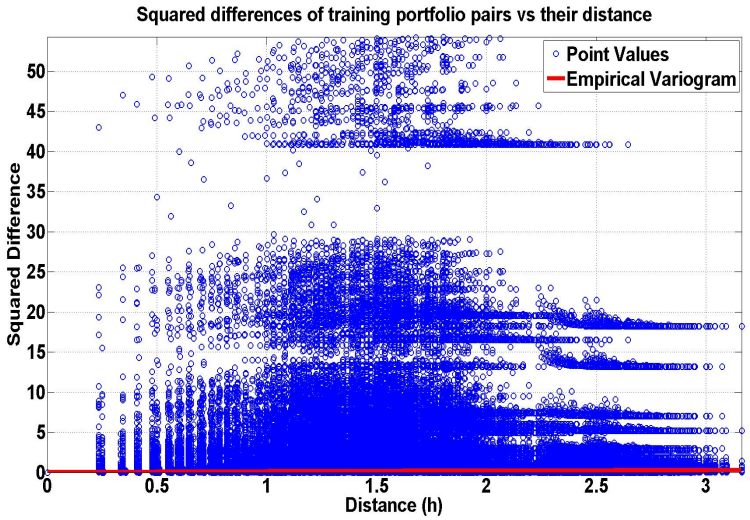


Figure 2.3: Squared difference of delta values of VA pairs in representative contracts.

are in close proximity to the empirical variogram and the variogram model. However, the data do not suggest the existence of any pattern from which a variogram model can be estimated. In particular, the data contradict the second-order stationary assumption underlying the Kriging method, and hence brings into question the appropriateness of the Kriging method for our application of interest.

Chapter 3

A Neural Network Approach to Estimation of Greeks

In Chapter 2, we provide numerical and theoretical results supporting our belief that a framework based on spatial interpolation [12] can successfully ameliorate the computational load of MC simulations by reducing the number of VA contracts that are subjected to nested MC simulation. However the proposed spatial interpolation framework requires an effective choice of distance function and a sample of VA contracts from the space in which the input portfolio is defined to achieve an acceptable accuracy level. The appropriate choice of the distance function for the given input portfolio in the proposed framework requires careful consideration by a subject matter expert for the given input portfolio. In this chapter[1], we propose to replace the conventional spatial interpolation techniques – Kriging, IDW and RBF [12] – in the spatial interpolation framework with a neural network. The proposed neural network can learn a good choice of distance function and use the given distance function to efficiently and accurately interpolate the Greeks for the input portfolio of VA contracts. The proposed neural network only requires knowledge of a set of parameters that can fully describe the types of VA contracts in the

[1]The material of this chapter is largely taken from [37]

input portfolio and uses these parameters to find a good choice of distance function.

3.1 Neural Network Framework

As we discuss in Chapter 2, spatial interpolation techniques can provide efficient and accurate estimation of the Greeks for a large portfolio of VA products. However, none of the traditional spatial interpolation techniques can provide us with all of accuracy, efficiency, and granularity. In particular, IDW and RBF methods provide better efficiency and resolution than Kriging methods, but they are less accurate than Kriging methods. Now, the question is whether there is a spatial interpolation technique that can be efficient, accurate and granular. However, before we try to answer this question, we should remind the reader that, as we discuss in Chapter 2, the choice of the representative contracts can significantly affect the accuracy of any interpolation scheme. In this chapter, assuming that a good choice of representative contracts exists, we want to discuss a choice of spatial interpolation scheme that can provide us with accuracy, efficiency, and granularity. We return to the question of a good choice of the representative contracts in Chapter 5.

Assuming $y(z_1), \cdots, y(z_n)$ are the observed values of the financial quantity of interest (e.g., a Greek) or approximations to them at locations $z_1, \cdots, z_n$, a spatial interpolation scheme provides an estimate $\hat{y}(z)$ of the quantity of interest at a location z where y is not known as a function of the values $D(z, z_1), D(z, z_2), \cdots, D(z, z_n)$, which are measures of similarity between the locations $z_1, z_2, \cdots, z_n$, and the location z, and the values $y(z_1), \cdots, y(z_n)$. Given the accuracy results of our numerical experiments for the models that we investigated in Chapter 2, we explore only the spatial interpolation schemes that are of the form $\hat{y}(z) = \sum_{i=1}^{n} w_i(D(z, z_1), \cdots, D(z, z_n))y(z_i)$ or $\hat{y}(z) = \sum_{i=1}^{n} w_i F(z - z_i)$, where $F(\cdot)$ is a radial function.

As we discuss in Chapter 2, IDW and RBF methods are the only methods we considered that can provide us with granularity. Unlike Kriging methods that solve an

optimization problem for each unknown location z to find the optimal choice of weight functions $w_i(\cdot), 1 \leq i \leq n$, RBF methods only do one optimization to find optimal choices of weights and IDW methods assume a particular shape for the functions $w_i(\cdot), 1 \leq i \leq n$, and hence do no optimization. Each of the optimization problems, either for the Kriging methods or the RBF methods, requires a time proportional to n^3. Such a time complexity does not scale well to a large portfolio of variable annuities, if Kriging is applied to each contract in the input portfolio, rather than the portfolio only. Therefore, if we want our spatial interpolation scheme to be granular, it should adopt a scheme similar to IDW methods or to RBF methods. In other words, it should either assume a particular shape for the functions $w_i(\cdot), 1 \leq i \leq n$, as IDW methods do, or it should solve one optimization problem, as RBF methods do, to find a global choice of weights and then uses these weights to do the estimation.

In what follows we use the description of a model in [5] to derive a spatial interpolation scheme with a structure similar to the structure of IDW methods that can provide all of the accuracy, granularity and efficiency. For additional details about this model, please refer to [5].

As we discuss in Chapter 1, VA products have complex structures and except for some simple VAs, like GMDB, and for simple risk metrics there exist no closed-form formula that determines the value of the key risk metrics of VA products. Therefore, no matter what method we use to value the key risk metrics of VA contracts, we can assume that the output of the method contains some errors. In other words, for VA contract z, we can assume that output value $y(z)$ can be written as

$$y(z) = y_t(z) + \epsilon$$

where $y_t(z)$ is the true value of the key risk metric of interest and ϵ is the error (noise or inaccuracy) in our estimation. Because MC simulations work under the supposition that a random process describes the evolution of the financial market and they choose a finite

number of realizations of this random process to determine their estimation of $y(z)$, we can assume that ϵ is non-deterministic. Therefore, we introduce the joint distribution $p(z, y(z))$. Now, assuming we have chosen the set of representative contracts $z_i, 1 \leq i \leq n$, we can use a Parzen density estimator to model the joint distribution $p(z, y(z))$ as follows.

$$p(z, y(z)) = \frac{1}{n} \sum_{i=1}^{n} f(z - z_i, y(z) - y(z_i)) \tag{3.1}$$

where $f(\cdot, \cdot)$ is the component density function [5]. We know that the best MSE estimate of $y^*(z)$ for a contract z is given by $\mathbb{E}[y(z)|z]$. If we now use equation (3.1), we have

$$\begin{aligned} y^*(z) = \mathbb{E}[y(z)|z] &= \int_{-\infty}^{\infty} y \times p(y|z) dy = \frac{\int y \times p(z, y) dy}{\int p(z, y) dy} \\ &= \frac{\sum_i \int y \times f(z - z_i, y - y_i) dy}{\sum_j \int f(z - z_j, y - y_j) dy} \\ &= \frac{\sum_i \int (t + y_i) \times f(z - z_i, t) dt}{\sum_j \int f(z - z_j, t) dt} \\ &= \frac{\sum_i y_i \times \left(\int f(z - z_i, t) dt \right) + \sum_i \int t \times f(z - z_i, t) dt}{\sum_j \int f(z - z_j, t) dt} \end{aligned} \tag{3.2}$$

where $y_i = y(z_i), 1 \leq i \leq n$. Now, if we assume that component density functions have zero mean[2] so that

$$\int_{-\infty}^{\infty} y \times f(z, y) dy = 0$$

for all values of z, then we can simplify equation (3.2) as follows.

$$y^*(z) = \frac{\sum_{i=1}^{n} G_H(z - z_i) \times y(z_i)}{\sum_{j=1}^{n} G_H(z - z_j)} \tag{3.3}$$

which is known as the Nadaraya-Watson model [54, 72]. This model is usually used in kernel regression applications and hence the function G_H in defined to be a kernel with

[2] In general, this is desired for fast convergence of an estimator to the true distribution [58]. Moreover, we are basically assuming that we are equally likely to over-estimate or underestimate the value of $y(z)$.

a bandwidth of H. The bandwidth H is not necessarily a scalar. It usually is a vector or a symmetric, positive definite matrix.

In general, for a d dimensional random vector z, as $n \to \infty$, the Parzen density estimator (3.1) converges to the true joint probability distribution of $p(z, y(z))$, if the function f in equation (3.1) satisfies the following sufficient conditions [58].

$$\begin{aligned} \sup_{(z,y)} |f(z,y)| &< \infty, \\ \int_{\mathbb{R}^{d+1}} |f(z,y)| dz dy &< \infty \\ \int_{\mathbb{R}^{d+1}} f(z,y) dz dy &= 1 \\ \lim_{||(z,y)|| \to \infty} ||(z,y)|| \times |f(z,y)| &= 0 \end{aligned} \tag{3.4}$$

If f satisfies (3.4) and is also a non-negative function, it is a probability distribution. In almost all practical applications, the function f is assumed to be a non-negative function that satisfies the conditions in (3.4). In this thesis, we also choose to make these assumptions about f. The G_H function in (3.3), is the marginal distribution of f and hence we expect it to have similar properties as f.

The choice of the G_H function and its bandwidth H implicitly define a distance function for the Nadaraya-Watson model. For example, if we choose G_H to be a Gaussian kernel, the distance function will have the form $D(z, z_i) \propto (\phi(z) - \phi(z_i))^T H^{-1} (\phi(z) - \phi(z_i)), 1 \leq i \leq n$, in which $\phi(z)$ determines the features (such as account value, guarantee value, age, etc.) that we use to describe the VA contract z. An example of features $\phi(z)$ are the vectors $\mathbf{x}$ and $\mathbf{y}$ in (2.10).

Equation (3.3) is similar, in structure, to equation (2.5) for the IDW estimator. In particular, the $G_H(z - z_i), 1 \leq i \leq n$, functions in equation (3.3) are comparable to the weights $w(D(z, z_i)), 1 \leq i \leq n$, in equation (2.5) of IDW. Once we know the choice of the G_H function and the H matrix, we can compute the Greeks for a large portfolio of VA contracts of size N in time proportional to $N \times n$ by calculating the Greeks for each

VA contract in the input portfolio and then summing the results. This preserves the efficiency and granularity of our framework. Considering the fact that equation (3.3) is a model for an optimal estimator (desirable for accuracy), we choose to use this model to develop our spatial interpolation technique.

A universal (location independent) choice of bandwidth for all locations of $z_i, 1 \leq i \leq n$, allows one to do conveniently the convergence analysis of Parzen density estimators. However, it also puts restrictions on the model of equation (3.3). As we discuss in Section 2.3.3, certain characteristics of a VA contract (e.g., moneyness (r) of the VA contract) are dependent on the location of VA contract in the space in which it is defined. These characteristics also allow one to explain certain trends in the value of key risk metric of interest. For example, as we discuss in Section 2.3.3, if moneyness of a contract is very small ($r \ll 1$) or very big ($r \gg 1$), the delta value of the contract is very close to zero and the changes in the delta value are minuscule. However, the delta value of contracts with $r \approx 1$ are significant. Therefore, to best capture the changes in the delta value when using the Nadaraya-Watson estimator, one should choose small bandwidth values for VAs with $r \approx 1$. However, for contracts with $r \ll 1$ or $r \gg 1$, we can choose larger bandwidth values. Motivated by this observation, in our research, we propose to use the following extended definition of the Nadaraya-Watson estimator:

$$\hat{y}(z) = \sum_{i=1}^{n} \frac{G_{h_i}(z - z_i) \times y(z_i)}{\sum_{j=1}^{n} G_{h_j}(z - z_j)} \tag{3.5}$$

where the subscript h_i, similar to the bandwidth H of kernels, denotes the range of influence of each $y(z_i)$ on the estimated value. Unlike the Nadaraya-Watson model, the h_is are not universal free parameters, but are location dependent.

Our experiments in Chapter 2 demonstrate the significance of the choice of the distance function on the accuracy of spatial interpolation techniques. A manual approach to find the best distance function that minimizes the estimation error for a given set of input data requires a search in the space of all distance functions which is not straight-

forward and hence requires investing a significant amount of time by a subject matter expert. Therefore, a manual approach to find the best distance function diminishes the effectiveness of the spatial interpolation techniques.

As we discuss above, once we determined the features that we will use to describe the VA contracts in the input portfolio, the choice of the G function and the $h_i, 1 \leq i \leq n$, values dictates the distance function in equation (3.5). Therefore, the optimal choice of distance function in a MSE sense can be defined as follows.

$$[G, h_1, \cdots, h_n] = \underset{G,h_1,\cdots,h_n}{\operatorname{argmin}} \frac{1}{2N} \sum_{j=1}^{N} ||\hat{y}(z_{I_j}) - y(z_{I_j})||^2 \tag{3.6}$$

in which $z_{I_j}, 1 \leq j \leq N$, are the VA contracts in the input portfolio and N is the size of the input portfolio. In equation (3.6), we have chosen to use $y(z)$ values instead of $y_t(z)$ values because the latter are not known to us; hence, we have to use the former, i.e., the outputs of the MC simulations, as the best estimates of the true values. The MSE is chosen as the objective function of the optimization problem (3.6) to allow us to have a fair comparison with the traditional spatial interpolation techniques discussed in [39]. The optimization problem (3.6) is well-defined if the minimum of the objective function is attainable. If we allow the $h_i, 1 \leq i \leq n$, to assume values on closed intervals only and consider a finite set of G functions that are non-negative and continuous on those intervals, we can guarantee that the optimization problem (3.6) is indeed well-defined. Notice that these assumptions, in practice, are not limiting assumptions. As we discussed earlier, the function G represents a marginal probability distribution. Therefore, it is reasonable to assume that it is non-negative. In practice, machines have limited precision which bounds the values that can be attained by $h_i, 1 \leq i \leq n$, values. The continuity of G with respect to the $h_i, 1 \leq i \leq n$, values and the finiteness of the set of G functions are sufficient conditions to guarantee that the minimum in (3.6) can be attained.

Solving the optimization problem (3.6) is not straightforward, even for a different choice of error function, and requires investing a significant amount of time. Therefore,

we should avoid a manual approach to find the best distance function to preserve the effectiveness of the spatial interpolation technique. We can decompose the optimization problem in (3.6) into two optimization problems: one for the choice of the G function and the other for the choice of the $h_i, 1 \leq i \leq n$, values.

$$\underset{G,h_1,\cdots,h_n}{\operatorname{argmin}} \frac{1}{2N} \sum_{j=1}^{N} ||\hat{y}(z_{I_j}) - y(z_{I_j})||^2 = \underset{G}{\operatorname{argmin}} \left(\underset{h_1,\cdots,h_n}{\operatorname{argmin}} \frac{1}{2N} \sum_{j=1}^{N} ||\hat{y}(z_{I_j}) - y(z_{I_j})||^2 \right) \quad (3.7)$$

For each choice of G function, the inner optimization problem in (3.7) can be automated using sub-optimal [3] iterative methods. As we discuss in Section 3.1.2, if the G functions are differentiable, an iterative method can use the gradients to substantially decrease the time to reach a sub-optimal solution. Our numerical experiments in Section 3.2 demonstrate that the calibration time accounts for the majority of the simulation time. Therefore, to reduce the calibration time, we propose to confine the choice of G to differentiable functions.

Gaussian, Epanechnikov, Trianglular, Quartic (Biweight), Triweight, Tricube, Cosine, and Sigmoid kernel functions all satisfy the constraints of (3.4), and are the most commonly used functions in kernel regression applications. Epanechnikov kernels are optimal with respect to Mean Integrated Squared Error (MISE) [28]. These functions have the common property that they are a nonlinear transformation of some polynomial function of their input. For these types of functions, there is a natural choice of feed-forward neural networks, described in the next section, that we can use to implement an iterative algorithm that computes an approximation of a local optimum of the bandwidth values. Therefore, we choose to focus on these classes of functions. Although this is an approximation to a local optimum, we demonstrate in our numerical results in this chapter that this choice can give us sufficient accuracy.

We discuss the implementation of the feed-forward network below and demonstrate

[3] A sub-optimal method converges to a local minimum while an optimal method converges to the global minimum.

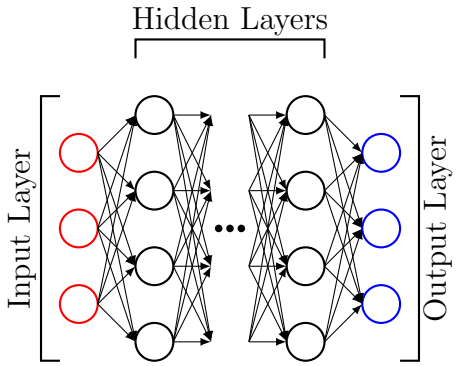


Figure 3.1: Diagram of a feed-forward neural network. Each circle represents a neuron.

its strong performance. For a thorough study of feed-forward networks, the interested reader is referred to [5] and the references therein. For the sake of brevity, in the rest of this thesis, we use the word neural network to refer to this particular class of feed-forward neural network unless explicitly said otherwise.

3.1.1 The Neural Network

A feed-forward neural network is a collection of interconnected processing units, called neurons, which are organized in different layers (Figure 3.1). The first and the last layers are respectively called the input layer and the output layer. Intermediate layers are called the hidden layers. Neurons of each layer take as input the outputs of the neurons in the previous layer. The neurons in the first layer serve only as inputs to the network. In other words, the neurons of the input layer produce what is known as the feature vector.

Assume $x_1, \cdots, x_s$ are the inputs of neuron j at hidden level l. First a linear combination of input variables is constructed at each neuron:

$$a_j^{(l)} = \sum_{i=1}^{s} w_{ij}^{(l)} x_i + b_j^{(l)} \tag{3.8}$$

where parameters $w_{ij}^{(l)}$ are referred to as weights and parameter $b_j^{(l)}$ is called the bias. The quantity $a_j^{(l)}$ is known as the activation of neuron j at level l. The activation $a_j^{(l)}$ is

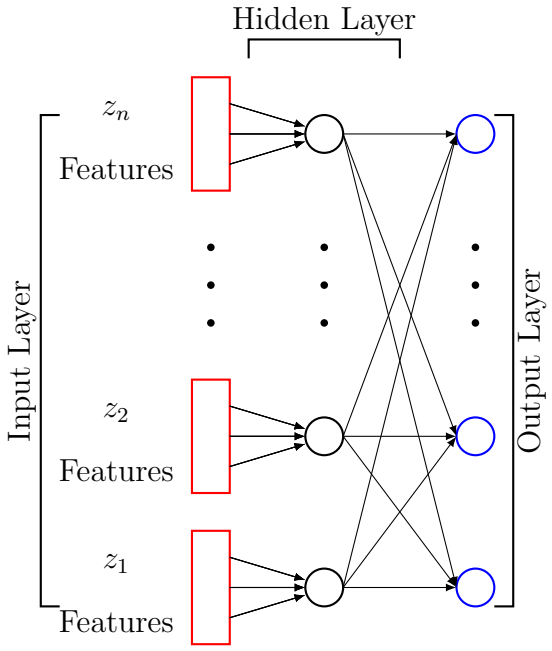


Figure 3.2: Diagram of the proposed neural network. Each circle represents a neuron. Each rectangle represent the set of neurons that contains input features corresponding to a representative contract.

then transformed using a differentiable, nonlinear function to give the output of neuron j at level l.

Let $\Phi(z) = (\phi_1(z), \phi_2(z), \cdots, \phi_d(z))$ represent the vector of features that are used to describe the VA contract z. Each feature $\phi_i(\cdot), 1 \leq i \leq d$, is a function of numeric attributes, or categorical attributes, or both and is defined at the discretion of a subject matter expert to explain the differences in the value of key risk metric of interest between different contracts. If we assume the inputs of a neuron in the hidden layer have the form[4] $\prod_{i=1}^{d}(\phi_i(z) - \phi_i(y))^{\alpha_i}$ or $\prod_{i=1}^{d}|(\phi_i(z) - \phi_i(y))|^{\alpha_i}$ in which $\alpha_i, 1 \leq i \leq d$, are non-negative integers, then we can see that the activation of this hidden layer is a polynomial of features of contracts z and y. Assuming the weight and bias parameters represent multiples of inverses of the bandwidths, if we use an appropriate transformation, e.g., exponential

[4]For example, if the G_{h_i} functions in (3.5) are Triangular or Tricube kernels, then the inputs of a neuron have this form.

function for a Gaussian kernel, the output of the hidden layer can represent the value of $G(z-y)$ for the class of G functions that we chose to use in the optimization problem (3.7). Therefore, we can use n of these neurons, one for each representative contract, to represent the $G_{h_i}(z-z_i), 1 \leq i \leq n$, functions in equation (3.5). Once, we have the outputs of this hidden layer, we can normalize these outputs in an output layer and use them to form the estimation of equation (3.5). Because of this natural representation, in our framework, we propose to use a neural network with only one hidden layer as in Figure 3.2.

As discussed above, the hidden layer in the proposed neural network contains one neuron for each $G_{h_i}, 1 \leq i \leq n$, function in equation (3.5). Hence, the number of neurons in the hidden layer corresponds to the number of representative contracts. The purpose of the output layer is to normalize the outputs of the neurons in the hidden layer. Therefore, for each neuron in the hidden layer there is a corresponding neuron in the output layer that normalizes the outputs of the corresponding neuron in the hidden layer. In other words, if the output of neuron i in the hidden layer represents $G_{h_i}(z-z_i)$ in equation (3.5), the output of the corresponding neuron in the output layer represents the term $\frac{G_{h_i}(z-z_i)}{\sum_{j=1}^{n} G_{h_j}(z-z_j)}$.

The inputs of each neuron in the hidden layer should be polynomials of the form $\prod_{i=1}^{d}(\phi_i(z)-\phi_i(y))^{\alpha_i}$ or $\prod_{i=1}^{d}|(\phi_i(z)-\phi_i(y))|^{\alpha_i}$ of the features $\phi_i(\cdot), 1 \leq i \leq d$, used to describe each VA contract. To achieve this, each neuron in the input layer will represent one such polynomial. We can have as many of these polynomial forms as necessary to get an accurate result. However, using too many of these polynomials results in data overfitting. One can use a validation portfolio, discussed in Section 3.1.2, to detect the data overfitting phenomena. Starting from polynomials of degree one, one can add polynomial forms of higher degrees in hope to see a decrease in the error in estimation of the key risk metric of interest for the validation portfolio. However after a certain point, adding further polynomial forms either does not decrease the estimation error

or, even worse, may increase it. It is at this point that we should stop adding further polynomials. In our experiments, with the class of G functions that provided the best performance results, polynomials of degree one were sufficient for good accuracy results, refer to Section 3.2. For that class of G functions, which we discuss in more detail below, we didn't get significantly better accuracy results by adding polynomials of higher degree. Therefore, to reduce the computational load, we choose to use only degree one polynomials $\phi_k(z) - \phi_k(y)$ or $|\phi_k(z) - \phi_k(y)|$ where $\phi_k(\cdot)$ is a feature used to describe a VA contract in the input portfolio.

For a feature $\phi(\cdot)$, define the positive direction around contract r as those contracts z for which $\phi(z) > \phi(r)$. We can correspondingly define the negative direction as contracts z for which $\phi(z) < \phi(r)$. Now consider the following linear combination of degree one polynomials $\phi(z) - \phi(r)$ and $|\phi(z) - \phi(r)|$ of feature $\phi(\cdot)$.

$$w_1(\phi(z) - \phi(r)) + w_2(|\phi(z) - \phi(r)|) \tag{3.9}$$

If we use degree one polynomials $\phi(z)-\phi(r)$ and $|\phi(z)-\phi(r)|$ as inputs of a neuron in the hidden layer that corresponds to the representative contract r, the linear combination (3.9) becomes part of the linear combination for the activation of that neuron. The weight parameters in equation (3.8) are the inverse of the bandwidth parameters. Thus, having linear combinations similar to (3.9) in the activation equation causes the value of the representative contract r to have a different range of influence in positive and negative directions around this contract. To see this difference, consider two contracts z_+ and z_- that are similar to representative contracts r except for the value of feature ϕ for which $\phi(z_+) - \phi(r) = \phi(r) - \phi(z_-)$. For these two contracts, the linear combination of (3.9) can be written as $(w_1 + w_2)|\phi(z_+) - \phi(r)|$ and $(w_2 - w_1)|\phi(z_-) - \phi(r)|$, respectively. These two linear combinations have different coefficients, unless $w_1 = 0$, which results in different activation values and subsequently different G values. Different G values in (3.5), in general, change the importance of $y(r)$ in the estimated value $\hat{y}(z)$.

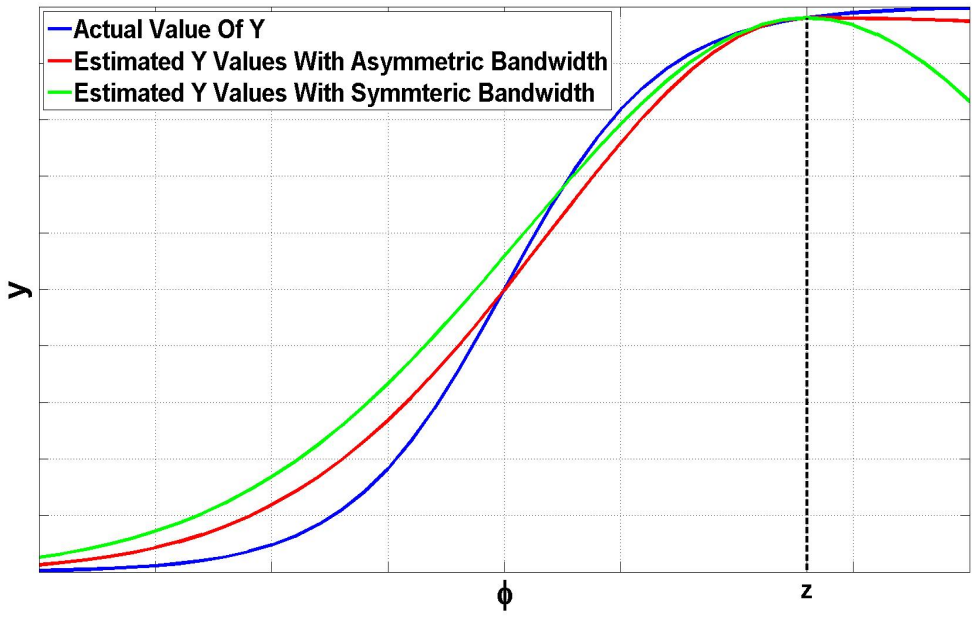


Figure 3.3: An example scenario in which having different bandwidth parameters in different direction around a representative contract can be beneficial.

There are scenarios in which having different bandwidths in different directions around representative contracts can allow for better estimations. For example, consider the scenario of Figure 3.3 in which z is a representative contract and ϕ is a feature. Moreover, assume that the value of $y(\cdot)$ is an increasing function of the value of $\phi(\cdot)$. For representative contract z, to prevent big underestimations in the positive direction of ϕ around z, we need the bandwidth parameter of ϕ in the positive direction to be big. However, to avoid big overestimations in the negative direction of ϕ around z, we need the bandwidth parameter to be small in the negative direction. To allow the neural network to properly deal with these scenarios, we propose for each representative contract and for each feature to use different bandwidth parameters in each direction around that representative contract. The red curve in Figure 3.3 shows how this asymmetric choice of bandwidths allows for better estimation of y values compared to the green curve in the same figure that uses similar choices of bandwidth in both directions around z. Notice that using

different bandwidth parameters in each direction around the representative contract is not a limiting assumption as it still allows symmetric bandwidth parameters to be defined for each feature. To allow the neural network to pick different bandwidth parameters for different directions of feature ϕ around a representative contract r, we propose to use $\max(\phi(z) - \phi(r), 0)$ and $\max(\phi(r) - \phi(z), 0)$ instead of the above mentioned first order degree polynomials as inputs to the neurons in the hidden layer.

Each VA contract is defined by the terms of the contract set up by the insurance company (such as the type of guarantee, the duration of contract etc.) and the characteristics of its policyholder (such as age, gender, etc.). We can use these characteristics to define risk indicators. For example, age itself is an indicator of the mortality risk. Also, the account value divided by the guarantee value defines the moneyness of the contract which is an indicator of the liability risk. Each risk metric can be defined as a function of a subset of these risk indicators. When valuing a risk metric, these risk indicators are what we refer to as features that describe a VA contract.

To this point, we considered features in terms of their numeric value. But some features used to describe VA contracts are categorical attributes (e.g., gender of the policyholder). These categorical attributes cannot be ignored. For example, IAM Mortality tables show that females have different mortality rates than males and hence their VA contracts are associated with a different mortality risk than males. Furthermore, our numerical experiments in Chapter 2 show that inclusion of categorical attributes can significantly affect the accuracy of spatial interpolation techniques. Therefore, we need to assign numeric values to these attributes so that we can incorporate them in our distance function. It is hard to imagine that a feature used to represent the VA contracts depends on a function of categorical attributes and numeric attributes. Therefore we propose to use each categorical attribute as a single feature to describe the VA contracts. If we simply assign a number to each category of a categorical attribute, we implicitly provide a measure of similarity between two categories. For example, consider a categorical at-

tributes with 3 categories c_1, c_2, c_3 and assume that we have assigned numbers 1, 2, and 3 to these categories, respectively. This implicitly assumes, for most distance functions, that c_1 and c_2 have more similar than c_1 and c_3. In categorical attributes that are often used to describe VA contracts, i.e., gender and rider, we didn't find any measure that can describe similarities of categories in the data. Moreover, in the literature, there is no discussion on the existence of such a measure. To avoid introducing non-existent similarities and considering that we are only interested in the difference of these categorical features, as we discuss above, we propose to consider the following function, known as overlapping similarity [6], to describe the difference between categorical attributes of VA contracts.

$$\phi_c(z) - \phi_c(y) = \begin{cases} 0 & \text{if } x_c(z) = x_c(y) \\ 1 & \text{if } x_c(z) \neq x_c(y) \end{cases} \tag{3.10}$$

where ϕ_c denotes a categorical attribute and $x_c(z)$ determines the category of that attribute for VA contract z. In general, we can choose any value v instead of 1, because these features are used as x values in equation (3.8). If h_1 is an optimal choice of bandwidth parameter for the given choice of function in (3.10), $\frac{h_1}{v}$ is an optimal bandwidth parameter for the choice of functions that uses v instead of 1 in (3.10). However, too large values of v can produce numerical instability by causing an overflow in the floating point variable used to store them on computers and too small values of v can slow-down the calibration (training) of the neural network. To avoid these problems, we normalize the features. Hence, we propose to use 1 in equation (3.10).

In the proposed neural network of Figure 3.2, the outputs of the neurons in the input layer provide the inputs of the hidden layer. Given our earlier discussions, the output of each neuron in the input layer represents a value in the set $\{F^c, F^-, F^+\}$. Each ϕ in F^c assumes the following form

$$f = \begin{cases} 0 & \text{if } x_c = x_{c_i} \\ 1 & \text{if } x_c \neq x_{c_i} \end{cases}$$

where x_c represents the category of categorical attribute c for input VA policy z, and x_{c_i} represents the category of categorical attribute c for representative VA policy z_i in the sample. Each value f in F^- has the form $f = [\phi(\mathbf{x}_{n_i}) - \phi(\mathbf{x}_n)]^+/R_\phi$, and each value f in F^+ has the form $f = [\phi(\mathbf{x}_n) - \phi(\mathbf{x}_{n_i})]^+/R_\phi$. In both of the aforementioned formulas, $\mathbf{x}_n$ is the vector containing the numeric attributes of input VA policy z, $\mathbf{x}_{n_i}$ is the vector containing the numeric attributes of the representative VA policy z_i in the sample, $\phi(\cdot)$ is a feature which, as we discuss earlier, determines a risk indicator using the numeric attributes of the contract. An expert user should determine the choice of the risk indicators that are suitable for the key risk metric of interest. The feature $\phi(\cdot)$ assumes a value in an interval of length R_t and $[\cdot]^+ = \max(\cdot, 0)$. As we discuss above, having two sets of f values, one in F^- and one in F^+, for each feature $\phi(\cdot)$ of each representative in the sample VA contract allows different bandwidths to be used in different directions around that representative VA contract. The term R_t is added for normalization purposes. If we don't normalize the features, the features that are defined on large intervals, e.g., account value, can create large weight updates (refer to Section 3.1.2) that may cause an overflow of floating point numbers used to store the corresponding weight parameters. Moreover, the features that are defined on small intervals, e.g., withdrawal rate, have small weight updates that may be too small, thus significantly increasing the training time. Therefore, to avoid these problems, we propose to normalize the values. Introducing the normalization factor R_t does not impose any constraints to the optimization problem of (3.7). If h_ϕ^+ and h_ϕ^- are the optimal choices of bandwidth parameters when $f_+ = [\phi(\mathbf{x}_n) - \phi(\mathbf{x}_{n_i})]^+$ and $f_- = [\phi(\mathbf{x}_{n_i}) - \phi(\mathbf{x}_n)]^+$ are used as x values in equation (3.8), we can attain the same optimum by using $f'_+ = [\phi(\mathbf{x}_n) - \phi(\mathbf{x}_{n_i})]^+/R_t$ and $f'_- = [\phi(\mathbf{x}_{n_i}) - \phi(\mathbf{x}_n)]^+/R_t$ as the x values in equation (3.8),

and $h_{\phi}^{+} \times R_t$ and $h_{\phi}^{-} \times R_t$ as the corresponding bandwidth parameters.

In our proposed neural network, neuron $i, 1 \leq i \leq n$, in the hidden layer is chosen to represent the value of the $G_{h_i}(z - z_i)$ function. The function $G_{h_i}(z - z_i)$ depends only on the features defined for the representative contracts z_i and input contract z. Therefore, we only need outputs of those neurons in the input layer that correspond to the values of f that are related to the representative VA policy i. Therefore, as shown in Figure 3.2, we propose to group the neurons in the input layer based on the representative contract that is used in the derivation of their outputs. Let group $i, 1 \leq i \leq n$, represent the neurons with outputs corresponding to the representative contract i. In the proposed neural network, the inputs of neuron i in the hidden layer are only connected to the outputs of neurons in group i of the input layer.

As we discuss earlier, the choice of the function that we use to transform activations in the hidden layer determines the family of kernel functions that we consider in the optimization problem (3.7). Our numerical experiments show that, except for exponential transformation of activations, the other choices of transformation function (i.e., cosine, polynomial, and sigmoid) result in gradient values that are either too small, which slows down the training process, or too big, which creates numerical instability by causing the floating point variables used in the implementation to overflow. Therefore, we propose that each neuron of the hidden layer transforms its activation using an exponential function to form its output.

To summarize, our proposed neural network allows us to rewrite equation (3.5) as

$$\hat{y}(z) = \sum_{i=1}^{n} \frac{\exp(\mathbf{w_i}^T \mathbf{f}(z, z_i) + b_i) \times y(z_i)}{\sum_{j=1}^{n} \exp(\mathbf{w_j}^T \mathbf{f}(z, z_j) + b_j)} \tag{3.11}$$

where the vector $\mathbf{f}(z, z_i)$ represents a vector of functions f, also called the feature vector in machine learning, from the set $\{F^c, F^+, F^-\}$ that are related to the representative VA policy z_i, and vector $\mathbf{w_i}$ contains the weights associated with each feature in $\mathbf{f}$ at neuron i of the hidden layer. Each $\mathbf{w_i}$ in (3.11) can be considered as the pointwise inverse of the

bandwidth value h_i in (3.5).

The condition that the integral of f equals 1 in (3.4) is a sufficient condition to guarantee that our Parzen density estimator is not biased. If we assume that all the weight and bias parameters in the proposed neural network are free parameters, none of the exponential functions in equation (3.11) represent a probability distribution. For each of the exponential functions in (3.11) to represent a probability distribution, the bias parameters $b_i, 1 \leq i \leq n$, should be functions of the weight parameters $\mathbf{w_i}$. This may at first glance seem like a deficiency of the method, but the end goal of the optimization problem (3.7) is to provide an accurate estimate of $y(z)$. To achieve that, we only need to provide accurate estimates of $p(y(z)|z)$ in (3.2). The $G_{h_i}, 1 \leq i \leq n$, functions provide estimates of $p(y, y(z))$, but the $\frac{G_{h_i}}{\sum_{j=1}^{n} G_{h_j}}$ functions provide estimates of $p(y(z)|z)$ and these normalized functions describe a probability distribution (i.e., they sum to one at each point). Therefore, we propose not to restrict the choice of the bias parameters and allow them to be free parameters so that we can have a bigger search space. As we demonstrate in our numerical results, this proposal does not introduce a significant bias in our estimations.

We can possibly extend the class of G functions that we consider in the optimization problem of (3.7) by adding extra hidden layers in the network. For example, the pdf of the log-normal distribution is not in the class of G functions that we chose to consider because it requires the natural logarithm of the features in its exponent. But a one-layer feed-forward network is capable of approximating any continuous function on a compact domain [40]. Therefore we can add a set of such one-layer feed-forward networks as an extra hidden layer to the network to approximate all the logarithmic functions required by the $G_{h_i}, 1 \leq i \leq n$, functions and feed the outputs of this added layer as inputs to the layer that represents $G_{h_i}, 1 \leq i \leq n$, functions.

A major problem with multi-layer neural networks, which we also found in our numerical experiments, is that, compared to 1 layer neural networks, multi-layer neural

networks require significantly more calibration time. In the problem that we are interested in, the calibration process of multi-layer neural networks requires supervised learning. As we discuss below, the fastest calibration processes require the computation of a gradient by a scheme called back-propagation. Back-propagation suffers from the problem of vanishing gradient which slows down considerably the calibration process of multi-layer neural networks [62]. Using additional layers also significantly increases the amount of computation that is required. Moreover, use of multi-layer networks requires us to determine the optimal number of neurons in each layer. Given the exploratory nature of our thesis and the strong performance results that we get with our proposed framework, we choose not to consider multi-layer neural networks in this thesis and leave them to be explored in future work.

3.1.2 Network Training Methodology

Equation (3.11) is a parametric formulation of our proposed estimator. As we discuss earlier, finding an analytical solution for the optimal choice of weight and bias parameters that minimizes the MSE error as defined in equation (3.6) is difficult. Therefore, we have to use sub-optimal iterative methods. Such iterative methods, first choose some initial guess $\mathbf{w}^{(0)}$ and $\mathbf{b}^{(0)}$ for the vectors of weight and bias parameters, respectively, and then search through the weight and bias parameters space in a succession of steps of form

$$[\mathbf{w}^{(t+1)}, \mathbf{b}^{(t+1)}] = [\mathbf{w}^{(t)}, \mathbf{b}^{(t)}] + [\Delta\mathbf{w}^{(t)}, \Delta\mathbf{b}^{(t)}] \tag{3.12}$$

where $\mathbf{w}^{(t)}$ and $\mathbf{b}^{(t)}$ denote the vectors of weights and bias parameters, respectively, at iteration t. The difference between various iterative methods is in the choice of their updated rule for computing the vector of updates, $[\Delta\mathbf{w}^{(t)}, \Delta\mathbf{b}^{(t)}]$, for the weight and bias parameters. This calibration process, in neural network literature, is known as network training.

The iterative methods can be divided into two major groups: non-gradient based

methods and gradient based methods. Non-gradient based methods are known for their high computational cost. The non-gradient based methods usually require more iterations than gradient based methods to reach a minimum and hence are poor choices of calibration process when efficiency is important [5]. Therefore, we choose to use gradient based methods to train the proposed neural network.

The gradient based methods either use only the gradient information to compute a weight/bias update vector or they also incorporate Hessian information. The gradient based methods that use Hessian information often require fewer iterations than the methods that use only gradient information to reach a local minimum. However, Hessian based methods normally have a higher computational cost per each iteration since they need to compute the Hessian matrix or an approximation to it [5]. The computational cost of Hessian based methods increases quadratically or cubicly in the length of the weight/bias vector. As we discuss in Section 3.2, the number of weight and bias parameters that we need to use in the proposed neural network is big. Therefore, to reduce the computational load of the training method, we choose to consider the methods that use only the information of the gradient, as is standard in most neural network training methods.

Amongst gradient based iterative methods that use only the gradient information, we choose to consider the family of gradient descent schemes that are known to provide the greatest rate of decrease in the error function [5]. The gradient descent scheme which is the simplest of these methods [7] updates the weight and bias parameters as follows

$$[\mathbf{w}^{(t+1)}, \mathbf{b}^{(t+1)}] = [\mathbf{w}^{(t)}, \mathbf{b}^{(t)}] - \eta \nabla E(\mathbf{w}^{(t)}, \mathbf{b}^{(t)}) \tag{3.13}$$

where the parameter $\eta > 0$ in (3.13) is known as the learning rate. The general convergence rate of gradient descent is $O(\frac{1}{T})$, where T denotes the number of iterations [67][5]. In this context, the rate of convergence is defined as the rate at which the

[5] f(x) = O(g(x)) means that there exist positive numbers c and M such that $\forall x > M : f(x) < cg(x)$

error, $|f(x_T) - f(x^*)|$, goes to zero, where f is a smooth convex function, x^* is the optimum value (value of interest) and x_T is the estimation of x^* after iteration T. The other methods in this family improve on the convergence rate of this method, under certain conditions, by optimizing the choice of the learning rate at each iteration.

Gradient descent methods, at each iteration, produce a higher rate of reduction in the directions of high-curvature than in the directions of lower-curvature. Big rate reductions in directions of high-curvature cause zig-zag movements around a path that converges to the local minimum and hence decrease the convergence rate [53]. However, a slower rate of reduction in directions of low-curvature allows for a persistent movement along the path of convergence to the local minimum. We can exploit this property by changing the weight update policy of gradient descent to use a velocity vector that increases in value in the direction of persistent reduction in the objective error function across iterations. This techniques is known as the momentum method [59].

The Nestrov's Accelerated Gradient (NAG) method [56], which can be considered as a variant of the classical momentum method [59] for general smooth convex functions and a deterministic gradient, achieves a global convergence rate of $O(\frac{1}{T^2})$ [67]. As we see below, the NAG method adds only a few computations to the equation (3.13); however, it significantly improves the convergence rate of the gradient descent method. The extra work added in this method is far less than the computational load introduced by other variants of the gradient descent scheme. Hence, to manage the computational load of the training method, we choose to use the NAG method to update our weight and bias parameters. In particular, we use a version of the NAG method described in [67] in which the NAG updates can be written as

$$\begin{aligned} v_{t+1} &= \mu_t v_t - \epsilon \nabla E([\mathbf{w}^{(t)}, \mathbf{b}^{(t)}] + \mu_t v_t) \\ [\mathbf{w}^{(t+1)}, \mathbf{b}^{(t+1)}] &= [\mathbf{w}^{(t)}, \mathbf{b}^{(t)}] + v_{t+1} \end{aligned} \tag{3.14}$$

where v_t is the velocity vector, $\mu_t \in [0,1]$ is known as the momentum coefficient and ϵ is the learning rate. In this scheme, the momentum coefficient is an adaptive parameter defined by

$$\mu_t = \min(1 - 2^{-1-\log_2(\lfloor \frac{t}{50} \rfloor + 1)}, \mu_{\max}) \tag{3.15}$$

where $\mu_{\max} \in [0,1]$ is a user defined constant. The proposed choice of the NAG method, compared to the original NAG method [56], uses an adaptive momentum coefficient to allow the method to apply to a broader class of functions.

As part of the weight updates in (3.14), we need to compute the gradient of the error function. But the gradient of the error function as described in (3.6) is dependent on the value of $y(z)$ for all the contracts in the input portfolio. Knowing $y(z)$ means that we have to do the MC simulations for all the contracts in the input portfolio which undermines why we are using this interpolation framework. Therefore we choose to approximate the error function in (3.6) by the MSE estimation error of a smaller set of VA policies which we call the training portfolio. The objective of the calibration process is then to find a set of weights and bias parameters that minimizes the MSE in the estimation of the key risk metric of interest (i.e., Greeks of the input portfolio in the rest of this chapter) for the training portfolio.

$$E(\mathbf{w}, \mathbf{b}) = \frac{1}{2|B|} \sum_{k=1}^{|B|} ||\hat{y}(\bar{z}_k, \mathbf{w}, \mathbf{b}) - y(\bar{z}_k)||^2 \tag{3.16}$$

where $\bar{z}_k, 1 \leq k \leq |B|$, are the VA policies in the training portfolio.

We choose the training portfolio to be different than the set of representative VA policies (i.e., observed points in the model (3.5)) to avoid data overfitting. Even with this choice of the training data, one cannot avoid the issue of overfitting. We discuss in Section 3.1.3 our solution to this problem.

Depending on the application of interest, the $y(\bar{z}_i)$ values can be too small (too big) resulting in too small (too big) gradient values for (3.16). Too small gradient values

increase the training time to reach a local minimum, while too big gradient values cause big jumps in the updates of (3.13) and hence numerical instability. Normalizing the values of $y(\bar{z}_i)$ in (3.16) and the choice of learning rate can help to ameliorate this problem[6].

The error function (3.16) uses the whole training set to compute the error function and subsequently the gradient of the error function in each iteration. Training techniques that use the whole training set in each iteration are known as batch methods [5]. Because of the redundancy in the data as well as the computational expense of evaluating gradients, batch gradient descent is a slow algorithm for training the network. Our experiments, further, corroborated the slowness of batch gradient descent in training our proposed network. To speed up the training, we used a particular version of what is known as the mini-batch training method [53]. In our training method, in each iteration, we select a small number (≤ 20) of training VA policies at random and train the network using the gradient of the error function for this batch. Hence, the error function in our mini-batch training method has the form

$$E(\mathbf{w^{(t)}}, \mathbf{b^{(t)}}) = \frac{1}{2|B^{(t)}|} \sum_{k \in B^{(t)}} ||\hat{y}(\bar{z}_k, \mathbf{w^{(t)}}, \mathbf{b^{(t)}}) - y(\bar{z}_k)||^2$$

where $B^{(t)}$ is the set of indices for selected VA policies at iteration t.

3.1.3 Stopping Condition

Figure 3.4 contains a graph of the MSE for a set of training VA policies as a function of the training iteration number for one run of our training algorithm. The graph, except at a few points, is a decreasing function of the iteration number which means that, as the iteration proceeds, the network is learning and steadily improving the bandwidth parameters for the model (3.11). After the first few thousand iterations, the graph of Figure 3.4 kneels and the rate of decrease in MSE drops dramatically. Such a significant

[6]Refer to the appendix for a discussion of heuristic ways to choose the free parameters described in this section.

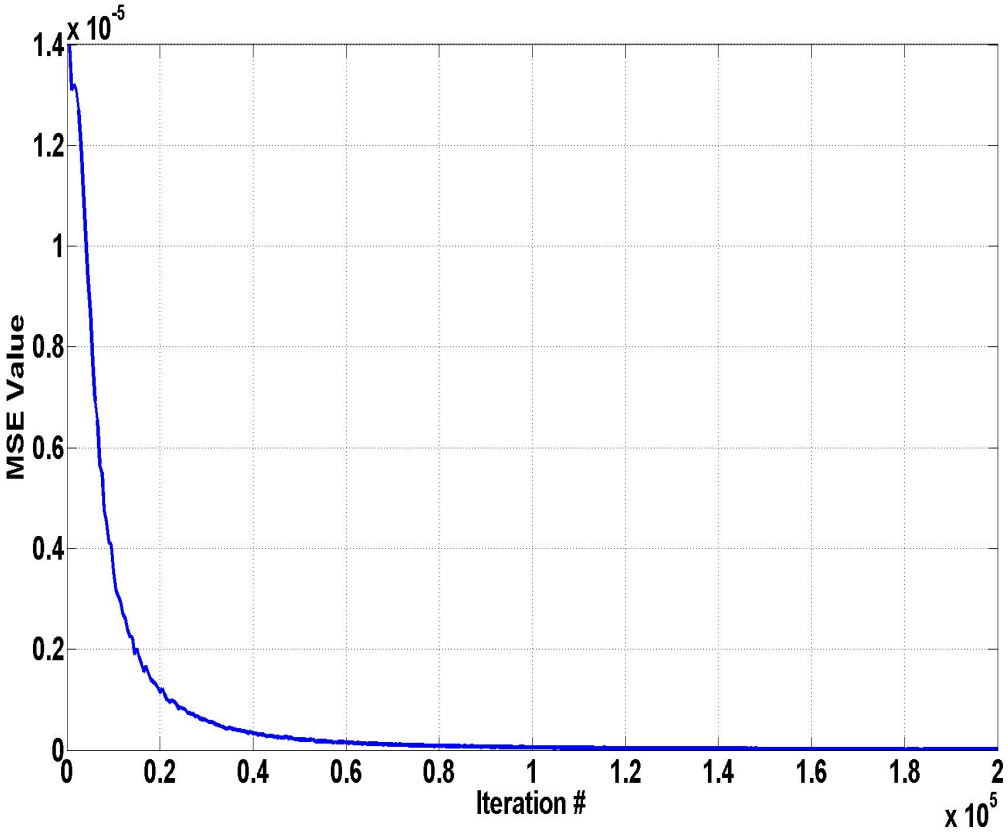


Figure 3.4: MSE of VA policies in the batch as a function of the iteration number.

drop in the rate of MSE reduction is usually a sign that the network parameters are close to their respective optimum values. If we train the network for a longer time, we expect the MSE to continue to decrease slowly. However, the amount of improvement in the accuracy of the network might not be worth the time that we spend in further training the network. Hence, it might be best to stop the training.

If we select VA policies for the training portfolio very close to the representative VA policies, training the network for a long time can cause data overfitting. Because a perfect solution for (3.11), in this case, is achieved when the bandwidth values tend to zero or equivalently the weight parameters become very large. However, such a network approximates the Greeks of VA policies that are not close to the representative VA policies by zero. To avoid over-training the network in such scenarios, we follow the

common practice in the machine learning literature and track the MSE for a set of VA policies which we call the validation portfolio [53]. The validation portfolio is a small set of VA policies that are selected uniformly at random from the VA policies in the input portfolio. The MSE of the validation set should decrease at first as the network learns optimal parameters for the model (3.11). After reaching a minimum value, the MSE of the validation portfolio often increases as the network starts to overfit the model (3.11) (Figure 3.5). In our training method, we propose to evaluate the MSE of the validation portfolio every I^{th} iteration of training, to avoid significantly slowing down the training process. We also propose to use a window of length W of the recorded MSE values for the validation set to determine if the MSE of the validation set has increased in the past $W-1$ recorded values after attaining a minimum. If we find such a trend, we stop the training to avoid overfitting. I and W are user defined (free) parameters and are application dependent.

As shown in the graph of Figure 3.5, the actual graph of the MSE for the validation portfolio as a function of iteration number might be volatile. However, a general u-shaped trend still exists in the data, which illustrates an increase in the value of the MSE after the MSE has reached a minimum. In order to find the trend graph, we use a simple moving average with a window size of $\bar{W}$ to smooth the data. We then fit, in the MSE sense, a polynomial of degree d to the smoothed data. We examine the resulting trend graph with windows of length W to determine the phenomenon of the MSE increase after attaining a local minimum. The parameters $\bar{W}$ and d are free parameters and are dependent on the application of interest.

So far, we have discussed two events, which we call stopping events, that can be used as indicators to stop the training. In both events, the network parameters are close to optimal network parameter values. At this point, each additional iteration of the training algorithm moves these parameters in a neighborhood of the optimal values and might make the network parameters closer to the optimal values or farther from the optimal

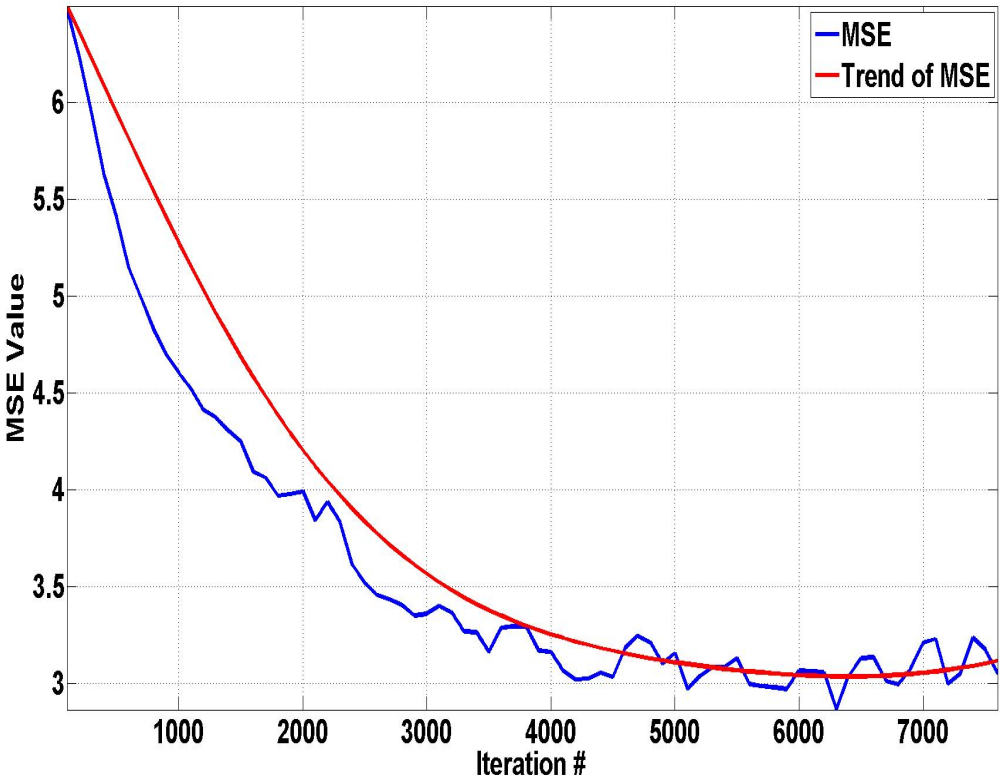


Figure 3.5: The MSE of the validation set and the trend in the MSE as a function of the iteration number for a run of the training algorithm. The trend is found using a moving average with a window size of 10 and then fitting a polynomial of degree 6 to the smoothed data.

values. Intuitively, the best time to stop the training is when the network parameters are very close to the optimal values and further training does not significantly improve them. In our training algorithm, we propose to use the relative error in an estimation of the mean of the Greeks of the validation portfolio as our stopping criteria. Let $\bar{G_{NN}}$ and $\bar{G_{MC}}$ denote the mean of the estimated Greeks for the validation portfolio computed by our proposed neural network approach and by MC simulations respectively. The relative error in estimation of the mean of the Greeks for the validation portfolio is then

$$\text{Err} = \frac{|\bar{G_{NN}} - \bar{G_{MC}}|}{|\bar{G_{MC}}|} \tag{3.17}$$

If the relative error (3.17) is smaller than a user defined threshold δ, we stop the training. The idea behind our choice of stopping criteria is that a good validation portfolio should be a good representative of the input portfolio. Hence, a network that has, on average, an acceptable accuracy in an estimation of the Greeks for the validation portfolio should, intuitively, have an acceptable accuracy in an estimation of the Greeks for the input portfolio as well. In some cases, finding stopping events and satisfying the stopping criteria may require the training algorithm to go through too many iterations, which can significantly increase the training time of the network and consequently decrease the efficiency of the method. We propose to stop the training algorithm once the network has gone through a user defined maximum number of iterations to limit the training time in such scenarios.

3.1.4 Sampling

As we discuss in Chapter 2, the choice of an appropriate sampling method is a key factor in obtaining an effective method within the proposed spatial interpolation framework. However, to focus on studying the effectiveness of our neural network approach in finding a good choice of distance function, and to keep the analysis simple, in this chapter, we use a very simple sampling method. We postpone a detailed discussion on the choice of a more effective sampling method to Chapter 5. However, in this section, we briefly describe ways in which the choice of our representative VA contracts can affect the performance of our proposed method.

Consider a realization of our proposed network with three representative contracts $x_1, x_2^{(1)}$ and x_3 with similar guarantee types. The VA contracts x_1 and $x_2^{(1)}$ are similar in every attribute except for the numeric attribute a_n and they differ with VA contract x_3 in every attribute. Now, consider another realization of our proposed network in which we

replace $x_2^{(1)}$ in the aforementioned realization with $x_2^{(2)}$. We choose $x_2^{(2)}$ such that it has similar categorical attributes as $x_2^{(1)}$; however, its numeric attributes assume the average of the corresponding numeric values for x_1 and x_3. Assume we train both networks for a similar number of iterations I. The gradient values of the error function depend only on the network architecture and the choice of input values. Since the input values for the corresponding hidden layer neurons for x_1 and $x_2^{(1)}$ in the former network are almost equal we expect the corresponding weight vectors $\mathbf{w_1^{(1)}}$ and $\mathbf{w_2^{(1)}}$ for these neurons to be approximately equal as well. However, because of the dissimilarity of the x_1 and $x_2^{(2)}$ contracts in the second network, we expect the input values and hence the corresponding weights $\mathbf{w_1^{(2)}}$ and $\mathbf{w_2^{(2)}}$ of the hidden layer neurons corresponding to these contracts to be quite different. Consequently, the latter network can provide a better differentiation between the x_1 and $x_2^{(2)}$ contracts while the former network requires more training time to provide the same level of accuracy in differentiating x_1 and $x_2^{(1)}$. Moreover, in approximating the Greeks for VA contracts other than $x_1, x_2^{(1)}, x_2^{(2)}$ and x_3, the former network, because of the similarity in weights $\mathbf{w_1^{(1)}}$ and $\mathbf{w_2^{(1)}}$, puts more emphasis on the corresponding Greeks of the contracts x_1 and $x_2^{(1)}$. Moreover, the latter network, because of the choice of $x_2^{(2)}$, can provide better accuracy for VA contracts that are quite different than both x_1 and x_3. Therefore, as demonstrated by this example, a bad sample can hurt the efficiency of the proposed method by requiring more training time. Moreover, a bad sample can hurt the accuracy of the proposed network in estimation of the Greeks of VA contracts that assume attribute values that are different than the representative contracts, in particular those VA contracts that are quite distant from any representative contract.

3.2 Numerical Experiments

In this section, we provide numerical results illustrating the performance of the proposed neural network framework in comparison with the traditional spatial interpolation

Attribute	Value
Guarantee Type	{GMDB, GMDB + GMWB}
Gender	{Male, Female}
Age	$\{20, 21, \ldots, 60\}$
Account Value	$[1e4, 5e5]$
Guarantee Value	$[0.5e4, 6e5]$
Withdrawal Rate	$\{0.04, 0.05, 0.06, 0.07, 0.08\}$
Maturity	$\{10, 11, \ldots, 25\}$

Table 3.1: GMDB and GMWB+GMDB attributes and their respective ranges of values for the synthetic input porttfolio.

schemes (i.e., Kriging, IDW, and RBF) discussed in [39]. The input portfolio in all experiments is a synthetic portfolio of $100,000$ VA contracts with attribute values that are chosen uniformly at random from the space described in Table 3.1. Similar to [39], we allow guarantee values to be different than the account values. The guaranteed death benefit of contracts with a GMWB rider is set to be equal to their guaranteed withdrawal benefit. The account values of the contracts follow a simple log-normal distribution model [42] with a risk free rate of return of $\mu = 3\%$, and volatility of $\sigma = 20\%$.

In our experiments, we use the framework described in [32] to value each VA contract. In each MC simulation, even in the calibration stage of the interpolation schemes to value representative contracts, we use $10,000$ scenarios. Fewer scenarios results in a noticeable difference, as big as 5%, between the computed delta value from successive runs. In our experiments, we use mortality rates of the 1996 IAM mortality tables provided by the Society of Actuaries.

We implement the framework in Java and run it on machines with dual quad-core Intel X5355 CPUs. We do not use the multiprocessing capability of our machine in these experiments; however, in Chapter 4, we demonstrate that even the serial implementation

of our proposed framework can provide better efficiency than parallel implementation of MC simulations.

3.2.1 Representative Contracts

As we discuss above in Section 3.1, to make our analysis more tractable, we do not address the issue of an effective sampling method in this chapter. Furthermore, in our experiments, the input portfolio, similar to Section 2.3, is a synthetic portfolio that is generated uniformly at random in the space of selected variable annuities. Hence, in all of the experiments in this section, we use a simple uniform sampling method similar to that in Section 2.3. The uniform sampling method generates sample portfolios that contain no duplicates VAs, and have sample VA contracts that are well-spaced to avoid issues discussed in Section 3.1.4. In each set of experiments, we select 300 representative contracts from the set of all VA contracts constructed from all combinations of points defined in Table 3.2. In a set of experiments, we select a set of representative contracts at the beginning of the experiment, and use the same set for various spatial interpolation methods that we examine in that experiment. This allows for a fair comparison between all methods.

3.2.2 Training/Validation Portfolio

Unlike traditional spatial interpolation schemes, we need to introduce two more portfolios to properly train our neural network. In each set of experiments, we select 250 VA contracts uniformly at random from the input portfolio as our validation portfolio.

For the training portfolio, we select 200 contracts uniformly at random from the set of VA contracts of all combinations of attributes specified in Table 3.3. The attributes of Table 3.3 are intentionally different from the attributes of Table 3.2 to avoid unnecessary overfitting of the data.

Attribute	Value
Guarantee Type	{GMDB, GMDB + GMWB}
Gender	{Male, Female}
Age	$\{20, 30, 40, 50, 60\}$
Account Value	$\{1e4, 1e5, 2e5, 3e5, 4e5, 5e5\}$
Guarantee Value	$\{0.5e4, 1e5, 2e5, 3e5, 4e5, 5e5, 6e5\}$
Withdrawal Rate	$\{0.04, 0.08\}$
Maturity	$\{10, 15, 20, 25\}$

Table 3.2: Attribute values from which representative contracts are generated for experiments.

Attribute	Value
Guarantee Type	{GMDB, GMDB + GMWB}
Gender	{Male, Female}
Age	$\{23, 27, 33, 37, 43, 47, 53, 57\}$
Account Value	$\{0.2e5, 1.5e5, 2.5e5, 3.5e5, 4.5e5\}$
Guarantee Value	$\{0.5e5, 1.5e5, 2.5e5, 3.5e5, 4.5e5, 5.5e5\}$
Withdrawal Rate	$\{0.05, 0.06, 0.07\}$
Maturity	$\{12, 13, 17, 18, 22, 23\}$

Table 3.3: Attribute values from which training contracts are generated for experiments.

3.2.3 Parameters of the Neural Network

In our numerical experiments, we use the following set of parameters to construct and train our network. We choose a learning rate of 1. We set $\mu_{\max}$ in (3.15) to 0.99. We use a batch size of 20 in our training. We fix the seed of the pseudo-random number generator that we use to select batches of the training data so that we can reproduce

our network for a given set of representative contracts, training portfolio, and validation portfolio. Moreover, we initialize our weight and bias parameters to zero.

The categorical features in F^c are rider type and gender of the policyholder. The following numeric features make up F^+.

$$f(z, z_i) = \frac{[\phi(x) - \phi(x_i)]^+}{R_\phi} \tag{3.18}$$

In our experiments, ϕ can assume the values maturity, age, AV, GD/AV, GW/AV, and withdrawal rate in which AV is the account value, GD is the guaranteed death benefit, and GW is the guaranteed withdrawal benefit. In Equation (3.18), R_ϕ is the range of values that ϕ can assume, x is the vector of numeric attributes for input VA contract z, and x_i is the vector of numeric attributes for representative contract z_i. The features of F^- are defined in a similar fashion by swapping x and x_i on the right side of Equation (3.18).

We record MSE every 50 iterations. We compute a moving average with a window of size 10 to smooth the MSE values. Moreover, we fit, in a least squares sense, a polynomial of degree 6 to the smoothed MSE values and use a window of length 4 to find the trend in the resulting MSE graph. In addition, we choose a δ of 0.005 as our threshold for the relative error in estimation of the Greeks for the validation portfolio.

3.2.4 Performance

In these experiments, we compare the performance (i.e., accuracy, efficiency, and granularity) of our proposed neural network scheme, referred to as NN in the results tables, with that of the traditional spatial interpolation schemes. From the set of interpolation techniques discussed in Chapter 2, we only choose the following interpolation methods with a corresponding distance function which exhibited the most promising results in Section 2.3.

- Kriging with Spherical and Exponential variogram models,
- IDW with power parameters p of 1 and 100,
- Gaussian RBF with free parameter ϵ of 1.

The distance function for the Kriging and RBF methods is

$$D(\mathbf{x}, \mathbf{y}, \gamma) = \sqrt{\sum_{h \in N} \left(\frac{x_h - y_h}{\max_h - \min_h} \right)^2 + \gamma \sum_{h \in C} \delta(x_h, y_h)}$$

where $N = \{\text{AV, GD, GW, maturity, age, withdrawal rate}\}$ is the set of numerical values and $C = \{\text{gender, rider}\}$ is the set of categorical values, and $\gamma = 1$.

For the IDW methods we choose the following distance function that provided the most promising results in Section 2.3.

$$D(\mathbf{x}, \mathbf{y}, \gamma) = \sqrt{f(x_{\text{age}}, y_{\text{age}}) g_{\text{age}}(\mathbf{x}, \mathbf{y}) + \sum_{h \in N} g_{\text{h}}(\mathbf{x}, \mathbf{y}) + \gamma \sum_{h \in C} \delta(x_h, y_h)}$$

$$f(x_{\text{age}}, y_{\text{age}}) = \exp\left(\frac{x_{\text{age}} + y_{\text{age}}}{2} - M \right)$$

$$g_h(\mathbf{x}, \mathbf{y}) = (\exp(-r_x) x_h - \exp(-r_y) y_h)^2$$

where $C = \{\text{gender, rider}\}$, $N = \{\text{maturity, withdrawal rate}\}$, $r = \frac{AV}{GD}$ and M is the maximum age in the portfolio.

Because we randomly select our representative contracts according to the method described in Section 3.2.1, we compare the performance of the interpolation schemes using 6 different realizations of the representative contracts, $Si, 1 \leq i \leq 6$. For our proposed neural network approach, we use the same training portfolio and validation portfolio in all of these 6 experiments. We study the effect of the training portfolio and the validation portfolio in a different experiment.

Although in this set of experiments and in many other experiments in this thesis we chose to report our findings using only a limited number of simulations (e.g., 5 or 6

simulations), this is not to suggest that we have conducted only these limited number of experiments to draw the conclusions that we present. Rather, for each set of experiments, we ran many simulations, and once we saw certain patterns emerge from the outcomes of those simulations, we reported only a few of these simulations that best represent these emerging patterns. For example, if we report the accuracy results of all of the experiments that we ran for the current set of experiments, the changes in mean and standard deviation of the relative error values, which we discuss in detail below, are very small. However, the general conclusions that we make by comparing the accuracy results of various methods still holds.

Table 3.4 displays the accuracy of each scheme in estimation of the delta value for the input portfolio. The accuracy of different methods is recorded as the relative error

$$\text{Err}_m = \frac{\Delta_m - \Delta_{MC}}{|\Delta_{MC}|} \tag{3.19}$$

where Δ_{MC} is the estimated delta value of the input portfolio computed by MC simulations and Δ_m is the estimate delta value of the input portfolio computed by method m. As we also discuss in Chapter 2, with $10,000$ inner loop scenarios, the Δ_{MC} of successive runs of MC simulations has a statistics of $\frac{\text{std}(\Delta_{MC})}{\text{mean}(\Delta_{MC})} = 0.25\%$.

The results of Table 3.4 show the superior performance of our proposed neural network (NN) framework in terms of accuracy. Except in a few cases, the accuracy of our proposed NN framework is better than all of the other interpolation schemes. Spherical Kriging has the best performance amongst the traditional interpolation schemes. Comparing the accuracy results of our proposed neural network scheme with that of Spherical Kriging shows that the relative error of the proposed scheme has lower standard deviation and hence is more reliable.

In Table 3.5, the average training and estimation time of each method is presented for two scenarios: (1) the method is used to estimate only the delta value of the entire portfolio and (2) the method is used to estimate the delta value of each policy in the

Method	Relative Error (%)					
	S1	S2	S3	S4	S5	S6
Kriging (Sph)	−0.60	0.55	7.62	2.86	2.58	3.59
Kriging (Exp)	−0.29	1.73	8.09	4.77	3.46	4.38
IDW (p = 1)	−21.43	−14.48	−21.76	−7.22	−12.47	−11.77
IDW (p = 100)	−11.74	−7.81	−4.36	−0.07	−2.72	−2.45
RBF (Gau, $\epsilon = 1$)	−0.76	−5.21	−10.45	−7.83	2.47	4.11
NN	−1.16	0.84	1.56	0.85	1.66	−1.46

Table 3.4: Relative error in estimation of the portfolio's delta value by each method.

input portfolio. Because of the complex calculations required to train the proposed NN method, the running time of the proposed NN method is longer than that of the traditional interpolation scheme. However, it still out performs the MC simulations (speed up of $> \times 15$).

In this experiment, assuming no prior knowledge of the market, we used zero as our initial value for weight/bias parameters, which is far from the optimal value and causes the performance of the proposed NN method to suffer from a long training time. In practice, insurance companies estimate the Greeks of their portfolios on frequent intraday basis to do dynamic hedging. Assuming a small change in the market condition, one does not expect the Greek values of the VA policies to change significantly. Hence, intuitively, the change in the optimal values of weight/bias parameters of the network under the previous and the current market conditions should be small. In Chapter 4, in the context of estimating the probability distribution of the one year loss, we demonstrate how we exploit this fact to reduce the training time of the network from an average of 4000 iterations to less than 200 iterations and hence reduce the training time significantly. In particular, assuming a neural network that has been calibrated to the previous market conditions, we construct a new network that uses the values of the weight/bias parameters

Method	Portfolio	Per Policy
MC	10617	10617
Kriging (Spherical)	41	$\gg 10617$
Kriging (Exponential)	41	$\gg 10617$
IDW (P = 1)	29	29
IDW (P = 100)	28	28
RBF (Gaussian, $\epsilon = 1$)	41	41
NN	539	539

Table 3.5: Simulation time of each method to estimate the delta value. All times are in seconds.

of the previous network as the initial values for the weight/bias parameters in the training stage.

A comparison of the running time in the two columns of Table 3.5 shows that the proposed NN method, similar to IDW and RBF, can be used to efficiently provide a granular view of the delta values in the input portfolio. Figures 3.6, 3.7, and 3.8 show a granular view of the estimated delta values of contracts in the input portfolio by our proposed NN scheme, the IDW method, and the RBF method, respectively. As we discuss earlier, Kriging methods cannot provide granularity and hence we cannot provide comparable graphs for them.

By comparing these graphs, we can see that the NN estimated delta values follow their corresponding MC estimated values (plotted data values are very close to the line $y = x$) much closer than the IDW and RBF estimated delta values do. In particular, the R^2 value of the $y = x$ line (the red line) for the proposed NN method is much closer to one than is the R^2 value for either the IDW and RBF methods. In fact, the graphs of Figures 3.7a and 3.7c show that IDW methods are not able to provide even modestly accurate estimates of delta values for contracts in the input portfolio. The R^2 value of

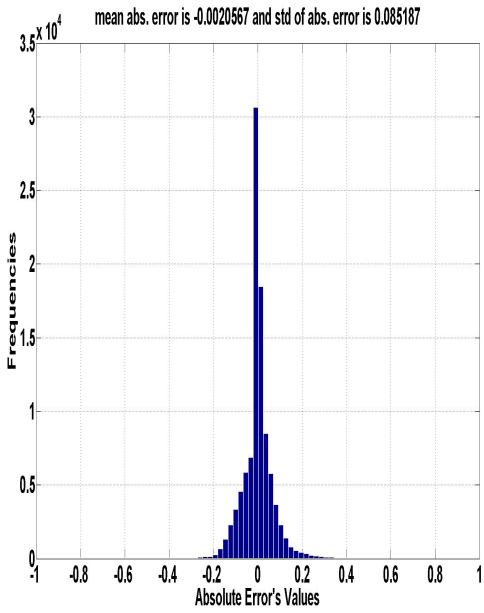
the line $y = x$ for the estimated delta values by the IDW methods is close to zero or even negative which shows very weak correlation between the estimated delta values by the IDW methods and the MC method.

Figures 3.6b, 3.7b, 3.7d, and 3.8b show the histogram of differences in estimation of delta values for contracts in the input portfolio via the MC simulations and each of the proposed NN methods, the IDW methods, and the RBF method, correspondingly. Amongst these interpolation methods, only the proposed NN method and the RBF method have histograms that are densely populated close to zero (i.e., low standard deviation in estimation error) and are symmetric around it (i.e., low bias in estimation error). However, the histogram of the proposed NN method is more concentrated around zero than the histogram of the RBF method. In fact, the statistics in the above-mentioned graphs show that the proposed NN method has the least mean and the least standard deviation of absolute estimation error amongst all the methods.

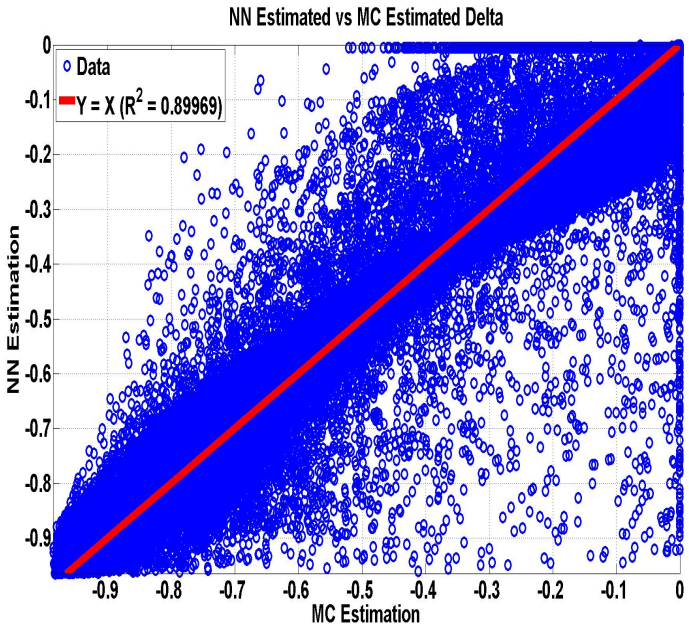
Although, the data of Figure 3.6a show large deviations from the line $y = x$, but the concentration of the histogram 3.6b around zero suggests that these deviations are in fact rare. Moreover, the symmetry of the histogram 3.6b further suggests the amount of over estimation by the neural network is close to the amount of under estimation by the neural network. Therefore, the estimation errors, in aggregate, cancel each other out, resulting in a smaller portfolio error.

3.2.5 Sensitivity to Training/Validation Portfolio

The training of our proposed NN method requires the selection of three VA portfolios. In the experiments of Section 3.2.4, we fix the selection of two of these portfolios (i.e., training portfolio and validation portfolio) while we measured the performance of our proposed method by changing the set of representative contracts. In the experiments of this section, we investigate the sensitivity of the accuracy and efficiency of our proposed method to the choice of training and validation portfolio. We conduct two sets of ex-



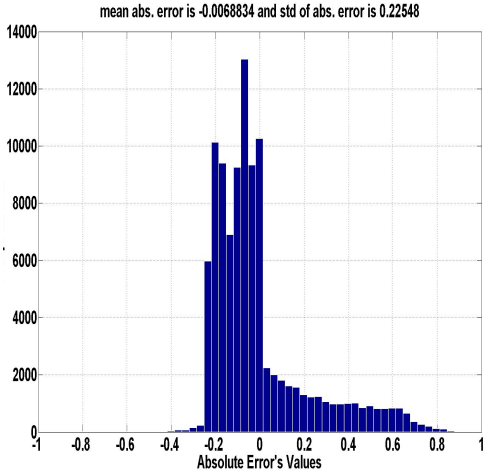
(a) Histogram of the difference in estimation of delta values for contracts in the input portfolio via the NN method and the MC simulations.



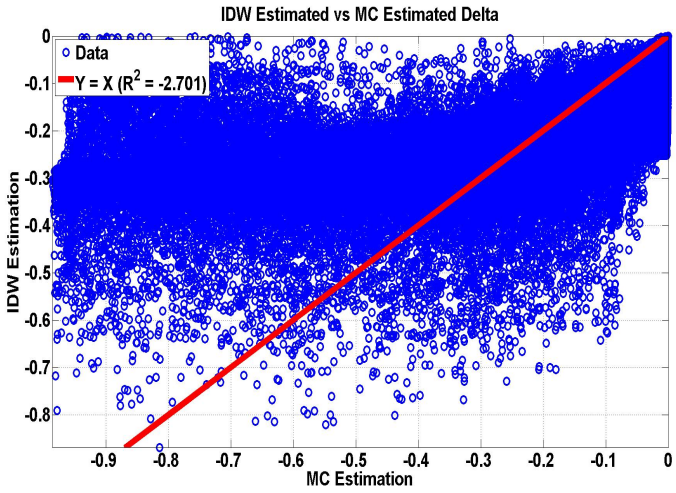
(b) Estimated delta values of contracts in the input portfolio by MC simulations and the NN method.

Figure 3.6: Comparing estimation of the delta values of the contracts in the input portfolio computed by the neutral network method and the MC method.

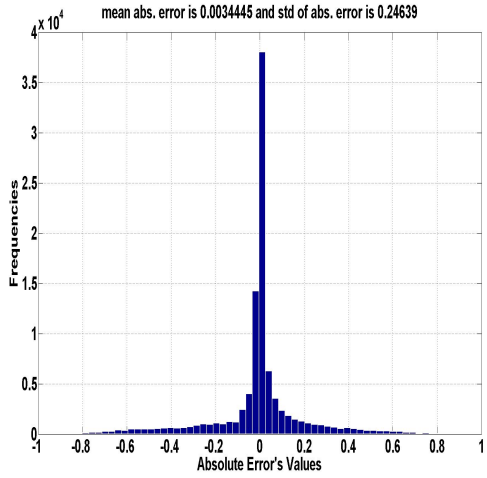
periments in which we fix the choice of representative contracts and either the training portfolio or the validation portfolio, while training the network with different realizations of the remaining portfolio. In the first set of experiments, we fix the choice of the representative contracts and the validation portfolio. We train the network with 5 different choices of the training portfolio and estimate the delta value of the input portfolio in each experiment. In the second set of experiments, we fix the choice of the representative contracts and the training portfolio and train the network with 5 different realizations of the validation portfolio. We then use the trained network to estimate the delta value of the input portfolio in each experiment. We used the same set of representative contracts in both set of experiments.



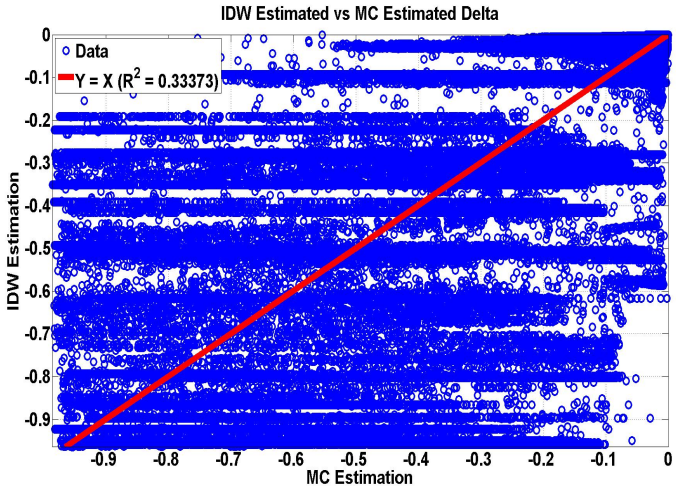
(a) Histogram of the difference in estimation of delta values for contracts in the input portfolio via the IDW ($P = 1$) method and the MC simulations.



(b) Estimated delta values of contracts in the input portfolio by MC simulations and the IDW method ($P = 1$).

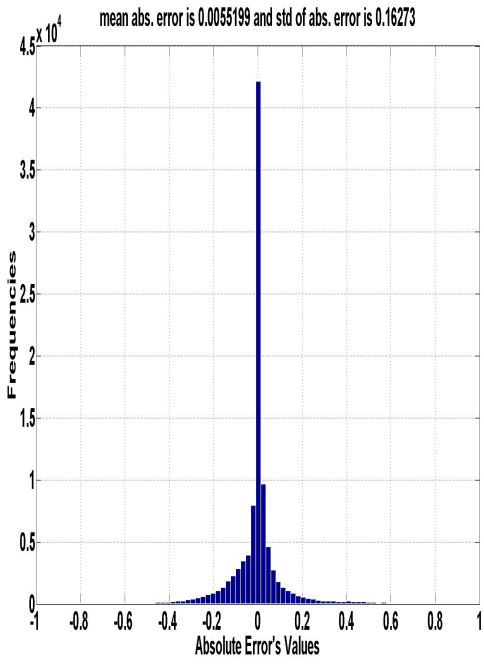


(c) Histogram of the difference in estimation of delta values for contracts in the input portfolio via the IDW method ($P = 100$) and the MC simulations.

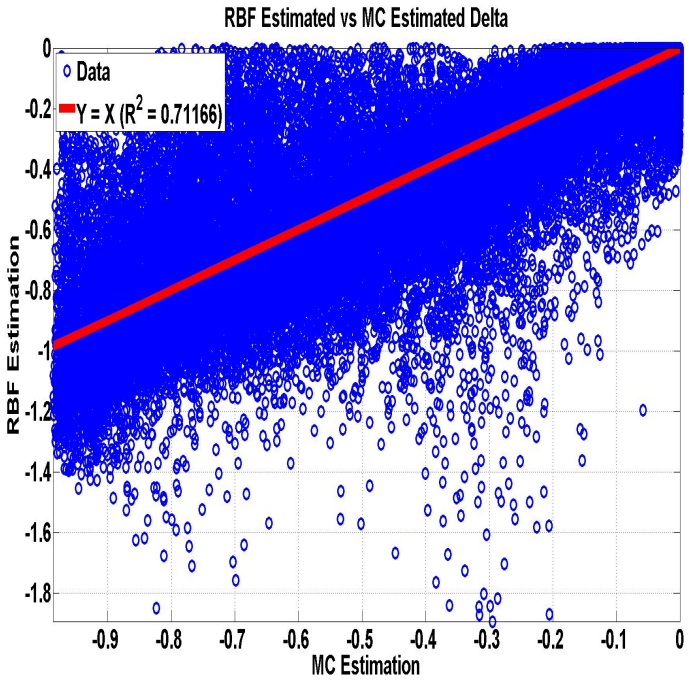


(d) Estimated delta values of contracts in the input portfolio by MC simulations and the IDW method ($P = 100$).

Figure 3.7: Comparing estimation of the delta values of contracts in the input portfolio computed by the IDW method and the MC method.



(a) Histogram of the difference in estimation of delta values for contracts in the input portfolio via the RBF method and the MC simulations.



(b) Estimated delta values of the contracts in the input portfolio by MC simulations and the RBF method.

Figure 3.8: Comparing estimation of the delta values of contracts in the input portfolio computed by the RBF method and the MC method.

Variable Portfolio	Relative Error (%)		Running Time	
	Mean	STD	Mean	STD
Training	0.27	1.52	660	246
Validation	−0.62	1.51	523	38

Table 3.6: Statistics on the running time sensitivity and accuracy sensitivity of the training network with different sets of training and validation portfolios. The recorded errors are relative errors as defined in (3.19). All times are in seconds.

The statistics for the running time [7] (training and estimation) and accuracy of each set of experiments are presented in Table 3.6. The relatively big values of the standard deviations indicate that the accuracy of the estimation is sensitive to the choice of the training portfolio and the validation portfolio. Despite this sensitivity, the method remains accurate.

The choice of the training portfolio can significantly affect the running time of the neural network; however, the running time of the network is fairly insensitive to changes in the choice of validation portfolio. The validation portfolio in the training is mainly used as a guard against overfitting. It is a useful stopping criteria to fine tune the network once we are close to the local optimum. On the other hand, the training portfolio defines the optimization problem that the training algorithm must solve. In particular, it determines how close the initial weight parameters are to the optimal choice of weight and bias parameters. For a fixed learning rate, a bad training portfolio can slow down the training by increasing the distance of the initial weight and bias parameters from the optimal choice of these parameters which necessitates more training iterations. Hence the choice of the training portfolio has a more significant effect than the choice of the validation portfolio on the running time.

3.2.6 Sensitivity to Sample Sizes

In the previous experiments, we examined the sensitivity of our proposed neural network framework to the selection of the training portfolio, the validation portfolio, and the set of representative contracts. In this section, we conduct experiments that assess the sensitivity of our proposed framework on the size of these portfolios. In each experiment, we fix two out of the three required portfolios while changing the size of the third portfolio. For each selected size of the latter portfolio, we train the network with 5 realizations of

[7] Although we fix the training portfolio or validation portfolio in these experiments, we still include the time that it took us to generate them in the reported running times.

Portfolio Sizes	Relative Error (%)		Running Time	
	Mean	STD	Mean	STD
$(300, 200, 250)$	0.38	1.35	539	120
$(250, 200, 250)$	-0.73	1.42	373	73
$(200, 200, 250)$	-1.62	1.52	310	85
$(300, 200, 250)$	0.27	1.52	539	246
$(300, 150, 250)$	-4.31	7.66	708	254
$(300, 100, 250)$	6.50	14.47	669	303
$(300, 200, 250)$	-0.62	1.51	523	38
$(300, 200, 200)$	0.70	3.23	511	24
$(300, 200, 150)$	2.31	3.67	582	188

Table 3.7: Statistics on running time sensitivity and accuracy sensitivity of training network with portfolios of various sizes. The recorded errors are relative errors as defined in (3.19). All times are in seconds.

the portfolio and record the running time and accuracy of the method.

Table 3.7 contains the statistics on the recorded running time and the relative error for each set of selected portfolio sizes. Each row in the table begins with a tuple (r, t, v) denoting the size of the set of representative contracts, the training portfolio, and the validation portfolio, respectively. In the scenarios corresponding to the first three rows, we changed the size of representative contracts. The second three rows show the results for the scenarios in which we changed the size of the training portfolio. Finally, in the scenarios corresponding to the third three rows, we changed the size of the validation portfolio.

The results of Table 3.7 show that decreasing the number of representative contracts increases the efficiency of the network. Furthermore, the amount of decrease in running

time is proportional to the amount of decrease in the number of representative contracts. This result is expected since the number of calculations in the network is proportional to the number of neurons in the hidden layer which is proportional to the number of representative contracts. Although the accuracy of the method deteriorates as we decrease the number of representative contracts, the accuracy of the worst network is still comparable to the best of the traditional spatial interpolation techniques (see Table 3.4). Hence, if required, we can sacrifice some accuracy for better efficiency.

According to the statistics in the second set of three rows of Table 3.7, decreasing the size of the training portfolio can significantly affect the accuracy of the method. Decreasing the number of training VA contracts results in a poorer coverage of the space in which the network is trained. In the space where the training VA contracts are sparse, the parameters of the representative VA contracts are not calibrated well, resulting in poor accuracy of estimation. Although the mean of the simulation time does not consistently decrease with the decrease in the size of the training portfolio, the standard deviation of the simulation time increases significantly. The increase in the standard deviation of the network's simulation time is a further proof that the network is struggling to calibrate its parameters for the smaller training portfolios.

The results in the last three rows of Table 3.7 suggest that decreasing the size of validation portfolio decreases the accuracy of the proposed framework. The deterioration in the performance of the network is more apparent from the amount of increase in the standard deviation of the relative error values. As one decreases the size of the validation portfolio, the VAs in the validation portfolio provide a poorer representation of the input portfolio. Although the change in the accuracy of the method is significant, the running time of the method is less affected by the size of the validation portfolio, except for the validation portfolio of the smallest size, where one can see a big increase in the standard deviation of the running time.

As we mentioned earlier in Section 3.2.5, the validation portfolio only affects the

last stage of the training where the network parameters are close to their local optimal values. Define the common neighborhood of a validation portfolio as the intersection between the δ neighborhood of the portfolio delta values for the validation portfolio and the local neighborhood of the optimal network parameter values. When the size of the validation portfolio is too small, various realizations of the validation portfolio may not adequately fill the space resulting in portfolio delta values that differ significantly from one realization to another. Hence, the common neighborhood of various realizations of validation portfolio may vary in place and size significantly. By definition, the stopping criteria terminates the training algorithm as soon as the training algorithm finds a set of network parameters that are within the common neighborhood of the validation portfolio. Therefore, the training time of the network can vary significantly based on the size and the place of the common neighborhood. As the size of the common neighborhood increases, the network spends less time searching for a set of network parameters that are within the common neighborhood. Because the training time is a significant part of the running time of the proposed neural network scheme, the standard deviation of the running time increases as a result of the increase in the standard deviation of the training time.

3.2.7 Sensitivity to the Size of Input Portfolio

In the experiments of this section, we investigate if the number of VAs in the input portfolio can affect the performance of the proposed NN framework. To keep the number of variable parameters small and hence have a better assessment, we keep the size of the representative portfolio, the training portfolio and the validation portfolio constant. We choose 300/200/250 as the size of the representative/training/validation portfolio because the NN framework in the previous experiments achieved the best performance results with these portfolio sizes.

The input portfolio in each set of experiment is a synthetic portfolio of VAs that is uniformly distributed in the space defined by attributes in Table 3.1. For each portfolio

Portfolio Size	Relative Error (%)		Running Time	
	Mean	STD	Mean	STD
$100,000$	0.38	1.35	539	120
$50,000$	0.58	2.99	681	158
$25,000$	-0.95	4.07	544	106

Table 3.8: Statistics on running time sensitivity and accuracy sensitivity of the neural network framework on input portfolios of various sizes. The recorded errors are relative errors as defined in (3.19). All times are in seconds.

size, we run 6 independent experiments each with a new set of representative portfolio, training portfolio and validation portfolio. The input portfolio is fixed for all 6 independent runs. The result of the experiments are recorded in Table 3.8.

From Table 3.8, we see that the accuracy of the NN framework decreases as the size of input portfolio decreases. In particular, the NN framework provides a poor estimation (relative to industry standards) for the input portfolio of size $25,000$. In fact, for the input portfolio of size $25,000$, we have to increase the parameter δ to 0.01 to be able to properly train the network and obtain the recorded results of Table 3.8.

We select the VAs of the representative portfolio to uniformly fill the space because we assume that the synthetic input portfolio is uniformly distributed in the space. Because the VAs of the input portfolio are randomly generated, the accuracy of the aforementioned assumption depends on the size of the input portfolio. The uniform distribution assumption is less accurate for the input portfolios of smaller size which causes a mismatch between the distribution of the input portfolio and the representative contracts. The mismatch between the distributions of the input portfolio and the validation portfolio leads to bias in the estimation of the portfolio delta value. The mean accuracy results of Table 3.8 show evidence of this bias. The mean accuracy results grow in absolute value as the size of the input portfolio decreases which indicates a growing bias.

Because the validation portfolio is a small sample of the input portfolio, the uniformity assumption is even less accurate for the validation portfolio. Hence, for small input portfolios, the representative portfolio may have a bigger bias in estimation of the portfolio delta value of the validation portfolio. Because of this bias, there may not be an overlap between the δ neighborhood of the portfolio delta value of the validation portfolio and the local neighborhood of the optimal values of weight and bias parameters in the second phase of training, i.e., the common neighborhood of the validation portfolio is empty. It is for this reason that we increased the value of δ to allow a bigger search space for the neural network to find a good choice of weight/bias parameters to finish the training. A better alternative approach to deal with this issue is to change the sampling method to consider the distribution of the input portfolio in the space. We discuss this alternative in more detail in Chapter 5.

The running time of the NN framework for the input portfolio of size $50,000$ is larger than that of the input portfolio of size $100,000$. Although NN framework's estimation time for the former has decreased, the training time of the NN framework for the input portfolio of size $50,000$ has increased compared to the training time of the NN framework for the input portfolio of size $100,000$. The increase in the training time is caused by the neural network's struggle to find an appropriate choice of weight/bias parameters in the second phase of the training. Increasing the value of parameter δ for the input portfolio of size $25,000$ alleviates this struggle and hence, as show in Table 3.8, results in a decrease in the training time and hence the running time of the NN framework.

Chapter 4

Application of Neural Network Framework in Estimation of SCR

The[1] Solvency II Directive is the new insurance regulatory framework within the European Union. Solvency II enhances consumer protection by requiring insurers to monitor the risks facing their organization. An integral part of Solvency II is the SCR that reduces the risk of insurers' insolvency. SCR is the amount of reserves that an insurance company must hold to cover any losses within a one year period with a confidence level of 99.5%.

The calculation standards are described in the documents of the Committee of European Insurance and Occupational Pensions Supervisors (CEIOP) (e.g., [16]). The regulation allows insurance companies to use either the standard formula or to develop an internal model based on a market-consistent valuation of assets and liabilities. Because of the imprecise language of the aforementioned standards, many insurance companies are struggling to implement the underlying model and to develop efficient techniques to do the necessary calculations. In [4, 21], rigorous mathematical definitions of SCR are provided. Moreover, [4] describes an implementation of a simplified, but approximately

[1]The material of this chapter is based on our paper [38].

equivalent, notion of SCR using nested MC simulations.

The results of the numerical experiments in [4] to find the SCR for a simple insurance product show that the proposed nested MC simulations are too expensive, even with their simplified notion of SCR. Hence, insurance companies cannot directly use the proposed MC approach to find the SCR for their large portfolios of insurance products. In this chapter, we propose a neural network approach to ameliorate the computational complexity of MC simulations which allows us to efficiently compute the SCR for large portfolios of insurance products. We provide insights into the efficiency of the proposed framework by studying its performance in computing the SCR for a large portfolio of VAs.

4.1 Solvency Capital Requirement

A rigorous treatment of SCR[2] requires the definition of Available Capital (AC) which is a metric that determines the solvency of a life insurer at each point in time. The AC is the difference between the Market Value of Assets (MVA) and Market Value of Liabilities (MVL):

$$\mathrm{AC}_t = \mathrm{MVA}_t - \mathrm{MVL}_t \tag{4.1}$$

where the subscript t denotes the time, in years, at which each variable is calculated.

Assuming the definition (4.1) of AC, the SCR, under Solvency II, is defined as the smallest amount of AC that a company must currently hold to insure a positive AC in one year with a probability of 99.5%. In other words, the SCR is the smallest amount x that satisfies the following inequality.

$$P(\mathrm{AC}_1 \geq 0 | \mathrm{AC}_0 = x) \geq 99.5\% \tag{4.2}$$

[2]The material in this section is based largely on the discussion in [4].

The application of definition (4.2) to find an approximate value of SCR, in practice, requires the estimation of the probability distribution of AC_1 for many AC_0 values (one probability distribution for each value of AC_0). Therefore, using definition (4.2) to find an accurate estimation of SCR is complex and is computationally intensive.

To reduce the computational complexity associated with (4.2), Bauer et al. use a simpler, approximately equivalent notion of the SCR which is based on the one-year loss function, Δ, evaluated at time zero:

$$\Delta = \mathrm{AC}_0 - \frac{\mathrm{AC}_1}{1+r} \tag{4.3}$$

where r is the one-year risk-free rate. The SCR is then redefined as the one-year Value-at-Risk (VaR)

$$\mathrm{SCR} = \mathrm{argmin}_x\{P(\Delta > x) \leq 0.5\%\} \tag{4.4}$$

Definition (4.4) allows one to find an estimation of SCR by estimating the probability distribution of Δ once only. We use the definition (4.4) of the SCR in the rest of this chapter.

4.2 Nested Simulation Approach

Given the definition (4.4) of SCR, we can calculate the SCR by first computing the empirical probability distribution of Δ and then computing the 99.5%-quantile of the calculated probability distribution. We can implement this scheme by the nested simulation approach of [4]. In this section, we first outline the nested simulation approach of [4] and then describe our modification of it to make it more computationally efficient.

In the nested simulation approach of [4], summarized in Figure 4.1, we first generate $N^{(p)}$ sample paths $\mathrm{P}^{(i)}, 1 \leq i \leq N^{(p)}$, that determine the one-year evolution of financial markets. Note that we are only interested in the partial state of the financial markets.

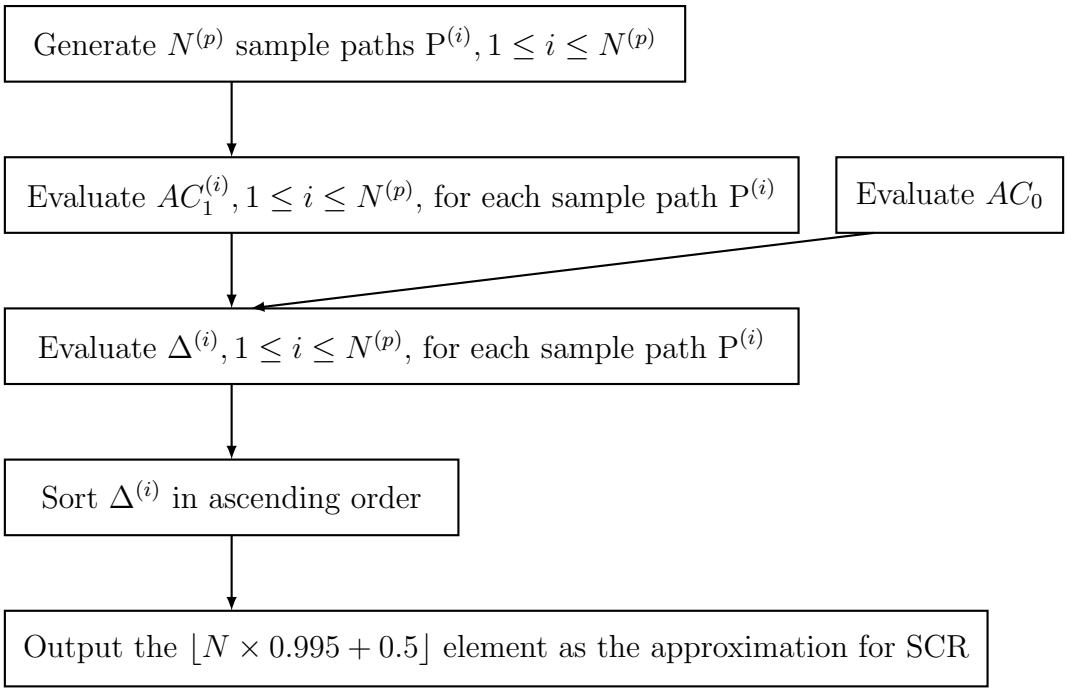


Figure 4.1: Diagram of the nested simulation approach proposed by [4].

In particular, we are only interested in the state of the financial instruments that help us evaluate the asset values and the liability values of our portfolio. Hence, we can generate a sample state of the financial market by drawing one sample from the stochastic processes that describe the value of those financial instruments of interest.

In the nested simulation approach of [4], for each sample path $\mathrm{P}^{(i)}$, we use a MC simulation to determine the value $\mathrm{AC}_1^{(i)}$, the available capital one year hence. We also calculate AC_0 via another MC simulation and use that to determine the value of $\Delta^{(i)}, 1 \leq i \leq N^{(p)}$, for each sample path $\mathrm{P}^{(i)}, 1 \leq i \leq N^{(p)}$, via equation (4.3). The values $\Delta^{(i)}, 1 \leq i \leq N^{(p)}$, can be used to determine the empirical distribution of Δ. In order to estimate the 99.5%-quantile for Δ as required by the definition of the SCR in equation (4.4), we sort the calculated $\Delta^{(i)}, 1 \leq i \leq N^{(p)}$, values in ascending order and choose the $\lfloor N \times 0.995 + 0.5 \rfloor$ element amongst the sorted values as the approximation for SCR.

The nested MC simulation approach of Figure 4.1 is computationally expensive even for simple insurance contracts [4]. The computational complexity of the approach stems

from two factors: 1) The value of $N^{(p)}$ can be very large [4]. 2) The suggested MC valuation of $\mathrm{AC}_1^{(i)}$ for each path $\mathrm{P}^{(i)}$ and AC_0, even for a single contract, is expensive and hence does not scale well to large portfolios of insurance products. In this thesis, we focus on the latter factor and provide an approach to significantly reduce the cost of computing each $\mathrm{AC}_1^{(i)}, 1 \leq i \leq N^{(p)}$, and AC_0. We also briefly discuss a proposal to address the former factor. However, we leave a detailed development and analysis of the proposal as a future work.

To begin, we briefly outline our proposal for reducing $N^{(p)}$ to address the first factor. In the nested simulation approach suggested by [4], to have a good estimation of the empirical probability distribution of Δ, the number of sample paths, $N^{(p)}$, must be large, since the $\Delta^{(i)}$ values, $1 \leq i \leq N^{(p)}$, are used to approximate the probability distribution of Δ. Consequently, many $\Delta^{(i)}$ values are needed to provide a sufficiently accurate approximation. Because a significant number of these values should, intuitively, be very close to each other, we suggest using a data interpolation scheme to reduce the cost associated with the number of sample paths, $N^{(p)}$. To do the interpolation, we first select/generate a small number, $N_s^{(p)}$, of paths $\mathrm{P}_s^{(i)}, 1 \leq i \leq N_s^{(p)}$, and evaluate $\Delta^{(i)}$ for each path $P_s^{(i)}$. Then, we use the calculated values $\Delta_s^{(i)}, 1 \leq i \leq N_s^{(p)}$, of the representative paths $\mathrm{P}_s^{(i)}, 1 \leq i \leq N_s^{(p)}$, to interpolate each $\Delta^{(i)}, 1 \leq i \leq N^{(p)}$, associated with each path $\mathrm{P}^{(i)}, 1 \leq i \leq N^{(p)}$. The choice of the interpolation scheme that should be used depends on the distribution of the generated $N^{(p)}$ paths in the space. The variables that define this space are dependent on the sources of randomness in the financial instruments that we use to value our portfolio. In this thesis, we use a simple linear interpolation scheme, described in more detail in Section 4.3, to reduce the running time of our numerical experiments. We postpone a more thorough development and analysis of the interpolation method as a future work.

Now we turn to the main focus of this chapter, a more efficient way to compute $\mathrm{AC}_1^{(i)}$ for each path $\mathrm{P}_s^{(i)}, 1 \leq i \leq N_s^{(p)}$, and AC_0. A key element in computing the Δ value via

equation (4.3) is the calculation of AC values. From (4.1), we see that the calculation of AC requires a market consistent valuation of assets and liabilities. Insurance companies can follow a mark-to-market approach to value their assets in a straightforward way. However, the innovative and complex structure of insurance products does not allow for such a straightforward calculation of liabilities. In practice, insurance companies often have to calculate the liabilities of insurance products by direct valuation of the cash flows associated with them (direct method [35]). Hence, the difficulty in calculation of SCR is primarily associated with the difficulty in the calculation of liabilities.

As we discuss in detail in Chapter 2, a MC simulation approach, as suggested in [4], to compute the liability of large portfolios of insurance products is very expensive. Furthermore, traditional portfolio valuation techniques, such as the replication portfolio approach [25, 26, 57] and the LSMC method [14, 15, 47], are not effective in reducing the computational cost.

As we discuss in detail in Chapter 2, a spatial interpolation approach can reduce the required computation of the MC scheme by reducing the number of contracts that must be processed by the MC method. In Chapter 3, we describe how a neural network approach to the spatial interpolation can not only solve the problem associated with finding a good distance metric for the portfolio but also provide a better balance between efficiency, accuracy, and granularity of estimation. The numerical experiments of Section 3.2 provided insights into the performance of our proposed neural network approach in estimation of Greeks for a portfolio of VAs. We show in this chapter how a similar neural network approach can be used to find the liabilities and subsequently the SCR for an input portfolio of VA products in an efficient and accurate manner.

Although we are using a neural network similar to the one proposed in Chapter 3, our experiments demonstrate that a blind usage of the neural network framework to estimate the liability values can provide no better computational efficiency than parallel implementation of MC simulations. As we discuss in Chapter 3, the time it takes to

train the neural network accounts for a major part of the running time of the proposed neural network framework. Therefore, if we have to train the network from scratch for each realization of the financial markets ($\mathrm{P}^{(i)}, 1 \leq i \leq N^{(p)}$) to compute $\Delta^{(i)}, 1 \leq i \leq N^{(p)}$, the proposed neural network loses its computational efficiency compared to a parallel implementation of the MC simulations. To address this problem, in Section 4.3, we discuss a methodology to use the parameters for a neural network trained to compute the $\mathrm{MVL}_1^{(i)}$ associated with $\mathrm{P}_s^{(i)}$ as a good first guess for the parameters for another neural network to compute $\mathrm{MVL}_1^{(j)}$ associated with $\mathrm{P}_s^{(j)}$, for $i \neq j$. The proposed methodology is based on the idea that, for two neural networks that are trained under two market conditions that are only slightly different, the optimal choices of neural network parameters are likely very close to each other.

In summary, we suggest to use the approach of Figure 4.2 instead of the nested simulation approach of Figure 4.1 to approximate the value of SCR via equation (4.4).

In this chapter, as mentioned earlier, our focus is on reducing the computational cost of the MC simulations used to calculate the liability of large portfolios of VAs. Hence, to focus on the problem of calculating the liabilities and to make the analysis more tractable, we assume that the company has taken a passive approach (i.e., no hedging is involved) and the only asset of the company is a pool of shareholders' money M_0 that is invested in a money market account and hence accrues risk-free interest. We understand that this is a very simple, and probably unrealistic, asset structure model; however, using a more complex asset structure only makes the computation of asset values more time consuming and diverts our attention from the key issue we are focusing on in this study, which is how to improve the efficiency of the computation of the liability values. Moreover, note that we can use the proposed framework to calculate the portfolio liability value of the input portfolio independently of the evaluation of asset values. That is, a more complex model for asset valuation can be inserted into our proposed framework without changing our scheme for improving the efficiency of the computation of liability values.

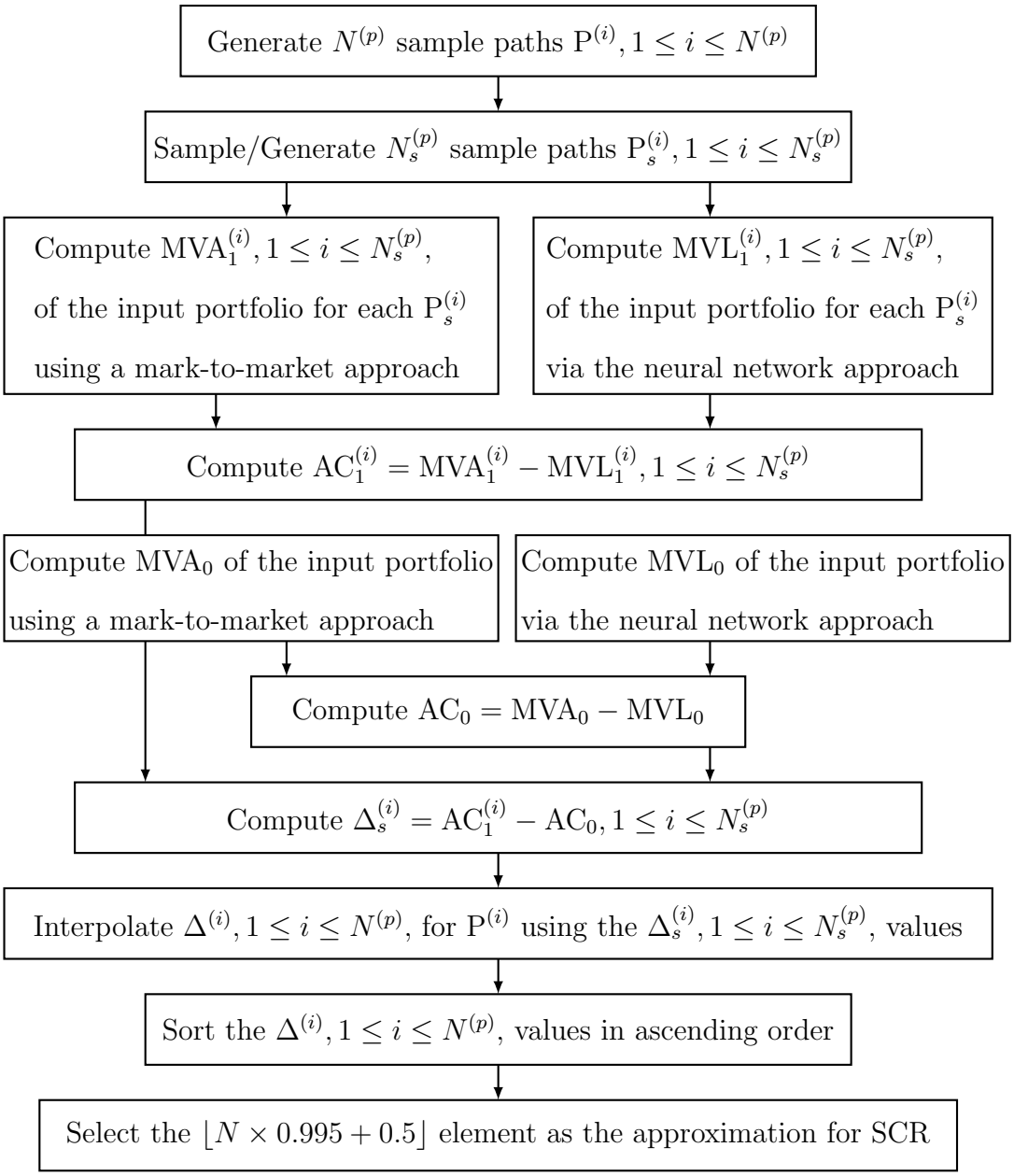


Figure 4.2: Diagram of the proposed nested simulation approach.

The proposed simple structure of assets allows us to eliminate the assets in the definition of Δ in (4.3) as follows.

Attribute	Value
Guarantee Type	{GMDB, GMDB + GMWB}
Gender	{Male, Female}
Age	$\{20, 21, \ldots, 60\}$
Account Value	$[1e4, 5e5]$
Guarantee Value	$[0.5e4, 6e5]$
Widthrawal Rate	$\{0.04, 0.05, 0.06, 0.07, 0.08\}$
Maturity	$\{10, 11, \ldots, 25\}$

Table 4.1: GMDB and GMWB attributes and their respective ranges of values.

$$\begin{aligned}\Delta &= \mathrm{AC}_0 - \frac{\mathrm{AC}_1}{1+r} \\ &= (M_0 - \mathrm{MVL}_0) - (\frac{(M_0(1+r) - \mathrm{MVL}_1)}{1+r}) \\ &= -\mathrm{MVL}_0 + \frac{\mathrm{MVL}_1}{1+r} \end{aligned} \tag{4.5}$$

Hence, in our simplified problem, calculating the SCR reduces to the problem of calculating the current liability and the distribution of the liability in one-year's time.

4.3 Numerical Experiments

In this section, we demonstrate the performance of the proposed neural network framework in calculating portfolio liability values using the nested simulation approach of Section 4.2. To do so, we estimate the SCR for a synthetic portfolio of $100,000$ VA contracts assuming the financial structure of assets as described in Section 4.2 that allows us to use equation (4.5).

Each contract in the portfolio is assigned attribute values uniformly at random from the space defined in Table 4.1. The guarantee values (death benefit and withdrawal

benefit) of GMWB riders are chosen to be equal[3], but they are different than the account value. The account values of the contracts follow a simple log-normal distribution model [42] with a risk free rate of return of $\mu = 3\%$, and volatility of $\sigma = 20\%$.

We acknowledge that this model of account value is very simple; we use it here to make our computations more tractable. A more complex model increases the number of MC simulations that is required to find liability values and affects the distribution of one-year-time's liability values. A more complex model of account values may also necessitate the use of more complicated valuation techniques for which the computational complexity is much more than simple MC simulations. However, regardless of the changes imposed by using a more complex model to describe the dynamics of the account value, the proposed nested simulation framework incorporating a neural network approach, described in Sections 4.2 and 3.1, can be used to calculate the SCR. Therefore, to focus on our neural network approach in this chapter, we have chosen to use a simple account value model to avoid distracting the reader with a lengthy description of a more complex account value model.

A change in the probability distribution of one-year-time's liability values changes the probability distribution of $\Delta^{(i)}, 1 \leq i \leq N^{(p)}$, values. A change in the probability distribution of $\Delta^{(i)}, 1 \leq i \leq N^{(p)}$, values can increase the number of sample $\mathrm{P}_s^{(i)}, 1 \leq i \leq N_s^{(p)}$, paths and the interpolation scheme that should be used to calculate the $\Delta^{(i)}, 1 \leq i \leq N^{(p)}$, values. As mentioned earlier, in this chapter, our focus is not on the choice of the interpolation scheme and/or the size $N_s^{(p)}$, the number of sample paths that should be used here. We leave these questions to future work. We can still repeat the experiments of this section and arrive at the same conclusions even if we directly compute the $\Delta^{(i)}, 1 \leq i \leq N^{(p)}$, values for the original $N^{(p)}$ paths $\mathrm{P}^{(i)}, 1 \leq i \leq N^{(p)}$; however, the running times will be much bigger.

We can increase the number of MC simulations or use more complex techniques to

[3]This is typical of the beginning of the withdrawal phase.

value liability values. But both of these approaches only slightly increase the running time of our proposed neural network approach to calculate liability values and hence only slightly increase the running time of our proposed nested simulation approach of Section 4.2. However the aforementioned approaches significantly increase the running time of the nested simulation approach of [4]. The nested simulation approach of [4] evaluates the liability values for each VA in the input portfolio; however, our proposed neural network framework only evaluates the liability values for the selected number of sample VA contracts of the representative portfolio, the training portfolio, and the validation portfolio and then does a spatial interpolation to find the liability values for the VAs in the input portfolio. Increasing the time to calculate the per VA liability value linearly increases the running time in the nested simulation approach of [4]. The size of the representative portfolio, the training portfolio and the validation portfolio combined in practice is much smaller than the size of the input portfolio. Therefore, increasing the time to calculate per VA liability only affects the total running time of the neural network to the extent that the calculation of liability values for the representative portfolio, the training portfolio, and the validation portfolio can affect the training time which in most of our experiments is not significant. Most of the computing time for the neural network approach is consumed in training the network.

We use the framework of [32] to value each VA contract. As in the previous chapters, we use $10,000$ MC simulations to value each contract. In our experiments, we use the mortality rates of the 1996 IAM mortality tables provided by the Society of Actuaries.

We implement our experiments in Java and run them on a machine with dual quad-core Intel X5355 CPUs. For each valuation of the input portfolio using the MC simulations, we divide the input portfolio into 10 sub-portfolios, each with an equal number of contracts, and run each sub-portfolio on one thread, i.e., a total of 10 threads, to value these 10 sub-portfolios in parallel. We use a similar parallel processing approach to value the representative contracts, the training portfolio and the validation portfolio.

Attribute	Value
Guarantee Type	{GMDB, GMDB + GMWB}
Gender	{Male, Female}
Age	$\{20, 30, 40, 50, 60\}$
Account Value	$\{1e4, 1e5, 2e5, 3e5, 4e5, 5e5\}$
Guarantee Value	$\{0.5e4, 1e5, 2e5, 3e5, 4e5, 5e5, 6e5\}$
Withdrawal Rate	$\{0.04, 0.08\}$
Maturity	$\{10, 15, 20, 25\}$

Table 4.2: Attribute values from which representative contracts are generated for experiments.

Although we use the parallel processing capability of our machine for MC simulations, we do not use parallel processing to implement our code for our proposed neural network scheme: our neural network code is implemented to run sequentially on one core. However, there is significant potential for parallelism in our neural network approach, which should enable it to run much faster. We plan to investigate this in future.

4.3.1 Network Setup

Although a sagacious sampling scheme can significantly improve the performance of the network, for the sake of simplicity, we use a simple uniform sampling method similar to that used in previous chapters. We postpone the discussion on the choice of a better sampling method to the next chapter. We construct a portfolio of all combinations of attribute values defined in Table 4.2. In each experiment, we randomly select 300 VA contracts from the aforementioned portfolio as the set of representative contracts.

As discussed in Section 3.1, in addition to the set of representative contracts, we need to introduce two more portfolios, the training portfolio and the validation portfolio, to

Attribute	Value
Guarantee Type	{GMDB, GMDB + GMWB}
Gender	{Male, Female}
Age	$\{23, 27, 33, 37, 43, 47, 53, 57\}$
Account Value	$\{0.2e5, 1.5e5, 2.5e5, 3.5e5, 4.5e5\}$
Guarantee Value	$\{0.5e5, 1.5e5, 2.5e5, 3.5e5, 4.5e5, 5.5e5\}$
Withdrawal Rate	$\{0.05, 0.06, 0.07\}$
Maturity	$\{12, 13, 17, 18, 22, 23\}$

Table 4.3: Attribute values from which training contracts are generated for experiments.

train our neural network. For each experiment, we randomly select 250 VA contracts from the input portfolio as our validation portfolio. The training portfolio, in each experiment, consists of 200 contracts that are selected uniformly at random from the set of VA contracts of all combinations of attributes that are presented in Table 4.3. In order to avoid unnecessary overfitting of the data, the attributes of Table 4.3 are chosen to be different than the corresponding values in Table 4.2.

We train the network using a learning rate of 20, a batch size of 20 and we set $\mu_{\max}$ to 0.99. Moreover, we fix the seed of the pseudo-random number generator that we use to select mini batches to be zero. For a given set of the representative contracts, the training portfolio, and the validation portfolio, fixing the seed allows us to reproduce the trained network. We set the initial values of the weight and bias parameters to zero.

We estimate the liability of the training portfolio and the validation portfolio every 50 iterations and record the corresponding MSE values. We smooth the recorded MSE values using a moving average with a window size of 10. Moreover, we fit a polynomial of degree 6 to the smoothed MSE values and use a window size of length 4 to find the trend in the MSE graphs. In the final stage of the training, we use a δ of 0.005 as our threshold for maximum relative distance in estimation of the liabilities for the validation

portfolio.

We use the rider type and the gender of the policyholder as the categorical features in F^c. The numeric features in F^+ are defined as follows.

$$f(z, z_i) = \frac{[\phi(x) - \phi(x_i)]^+}{R_\phi} \tag{4.6}$$

In our experiments, ϕ can assume the values maturity, age, AV, GD, GW and withdrawal rate, R_ϕ is the range of values that ϕ can assume, x and x_i are vectors denoting the numeric attributes of the input VA contract z and the representative contract z_i, respectively.

We define the features of F^- in a similar fashion by swapping x and x_i on the right side of equation (4.6).

4.3.2 Performance

The experiments of this section are designed to allow us to compare the efficiency and the accuracy of the proposed neural network approach to the nested simulations with the nested MC simulation approach of [4]. In each experiment, we use $N^{(p)} = 40,000$ realizations of the market to estimate the empirical probability distribution of Δ. As we describe in Section 4.2, the particular simple structure of assets that we use allows use to use equation (4.5) to evaluate Δ. By design, the liability value of the VA products that we are using in our experiments is dependent on their account values. As mentioned earlier, the account values follow a log-normal distribution model. Hence, we can describe the state of the financial market by the one-year's time output of the stochastic process of the model. Assuming a price of A_0 as the current account value of a VA, each realization of the market corresponds to a coefficient C_1, from the above-mentioned log-normal distribution, that allows us to determine the account value in one year's time as $A_1 = C_1 \times A_0$.

To come up with sample paths $\mathrm{P}_s^{(i)}$, we determine a range (interval) based on the

maximum value and the minimum value of the generated $40,000$ $C_1^{(i)}, 1 \leq i \leq 40,000$, coefficients that describe the state of the financial markets in one-year's time and divide that range into 99 equal length sub-intervals. We use the resulting 100 end points, $C_{s_1}^{(i)}, 1 \leq i \leq 100$, as the sample paths $\mathrm{P}_s^{(i)}, 1 \leq i \leq 100$.

If one graphs the resulting $\Delta_s^{(i)}, 1 \leq i \leq 100$, values as a function of the $C_{s_1}^{(i)}, 1 \leq i \leq 100$, values that describe the evolution of the financial markets, the resulting curve is very smooth and the 100 points are very close to each other in the space. Because of this, we chose to interpolate the value of $\Delta^{(i)}, 1 \leq i \leq 40,000$ for the aforementioned $40,000$ realizations of the financial markets $(C_1^{(i)}, 1 \leq i \leq 40,000,)$ by a simple piecewise-linear interpolation of the $\Delta_s^{(i)}, 1 \leq i \leq 100$, values. As we discuss later, the choice of a piecewise-linear interpolator might not be optimal. As noted earlier in Section 4.2, we use it here as a first simple choice for an interpolation. We plan to study the choice of possibly more effective interpolations later.

To estimate the liability values for each of the sample paths $\mathrm{P}_s^{(i)}$ via the proposed neural network framework, we first generate the representative portfolio, the training portfolio, and the validation portfolio. We then train the network using the liability of values at time 0 (current liability) of VAs in these portfolios. We use the trained network to estimate the liability of the input portfolio at time 0.

As we mention in Section 4.2, if we train the network before estimating each liability, the running time of the proposed neural network approach, because of the significant time it takes to train the network, is no better than a parallel implementation of MC simulations. To address this issue, we use the above-mentioned trained network for the liability values of times 0 to estimate the one year liability of the input portfolio for each end point, $C_{s_1}^{(i)}, 1 \leq i \leq 100$. However, before each estimation, we perform the last stage of the training method to fine-tune the network. More specifically, we train the network for a maximum of 200 iterations until the network estimated portfolio liability for the validation portfolio is within $\delta = 0.01$ relative distance of the MC estimated portfolio

liability of the validation portfolio. If the fine-tuning of the network is unable to estimate the liability of the validation portfolio within the defined δ relative distance, we define a new network using the set of representative contracts, the training portfolio and the validation portfolio and train the new network– i.e., we do the complete training. We then use the new trained network in the subsequent liability estimation– i.e., we use the new trained network to do the fine-tuning and portfolio liability estimation for subsequent $C_{s_1}^{(i)}$ values.

The idea behind the above-mentioned proposal to reduce the training time is that if two market conditions are very similar, the liability values of the VAs in the input portfolio under both market conditions should also be very close and hence the optimal network parameters (weight parameters and bias parameters) for both markets are likely very close to each other as well. If the optimal network parameters are indeed very close, the fine-tuning stage allows us to reach the optimal network parameters for the new market conditions without going through our computationally expensive training stage that searches for the local minimum in the whole space. If the fine-tuning stage fails, then we can conclude that the local minimum has changed significantly and hence a re-training in the whole space is required.

To effectively exploit the closeness of market conditions to reduce the training time, we have to sort $C_{s_1}^{(i)}, 1 \leq i \leq 100$, values and evaluate the portfolio liability values in order. Otherwise, we might have a scenario in which consecutive values of $C_{s_1}^{(i)}$ represent market conditions that are not relatively close. Under such a scenario, the fine-tuning stages will most likely fail requiring us to do a complete training of the neural network for each $C_{s_1}^{(i)}, 1 \leq i \leq 100$. The above-mentioned proposal is not as straightforward for more complex models in which the dynamics of financial markets is described with more than one variable. The choice of an effective strategy to exploit the closeness of market conditions to reduce the training time for more complex models of financial markets requires further investigation and we leave it as a future work.

We compare the performance of the interpolation schemes using 6 different realizations, $S_i, 1 \leq i \leq 6$, of the representative contracts, the training portfolio, and the validation portfolio. Table 4.4 lists the accuracy of our proposed scheme in estimating MVL_0, the 99.5%-quantile of MVL_1 ($\text{MVL}_1^{(99.5)}$), which corresponds to the 99.5%-quantile of the Δ, and the SCR value for each scenario. Accuracy is recorded as the relative error

$$\text{Err} = \frac{X_{NN} - X_{MC}}{|X_{MC}|} \tag{4.7}$$

where X_{MC} is the value of interest (liability or the SCR) in the input portfolio computed by MC simulations and X_{NN} is the estimation of the corresponding value of interest computed by the proposed neural network method.

The results of Table 4.4 provide strong evidence that our neural network method is very accurate in its estimation of MVL_0 and $\text{MVL}_1^{(99.5)}$. The estimated liability values also result in very accurate estimation of the SCR, except for scenarios S_4 and S_5. Even for scenarios S_4 and S_5, the estimated SCR values are well within the desired accuracy range required by insurance companies in practice. Our numerical experiments in Section 3.2 show that our proposed neural network framework has low sensitivity to the particular realization of the representative contracts, and the training/validation portfolio once the size of these portfolios are fixed. The results of Table 4.4 further corroborate our finding in Chapter 3 as the realization of the representative contracts, and the training/validation portfolio is different in each scenario.

The accuracy of the proposed method can be further examined by considering Figure 4.3 in which the estimated liability values, for the 100 end points of the intervals, by the proposed neural network method are compared with their respective MC estimations. The graphs of Figures 4.3b and 4.3c show that the liability estimated values by the proposed neural network method are very close to those of the MC method, which demonstrates the projection capabilities of the neural network framework. As we discuss earlier, the smoothness of the MC liability curve, as shown in Figure 4.3, motivated us

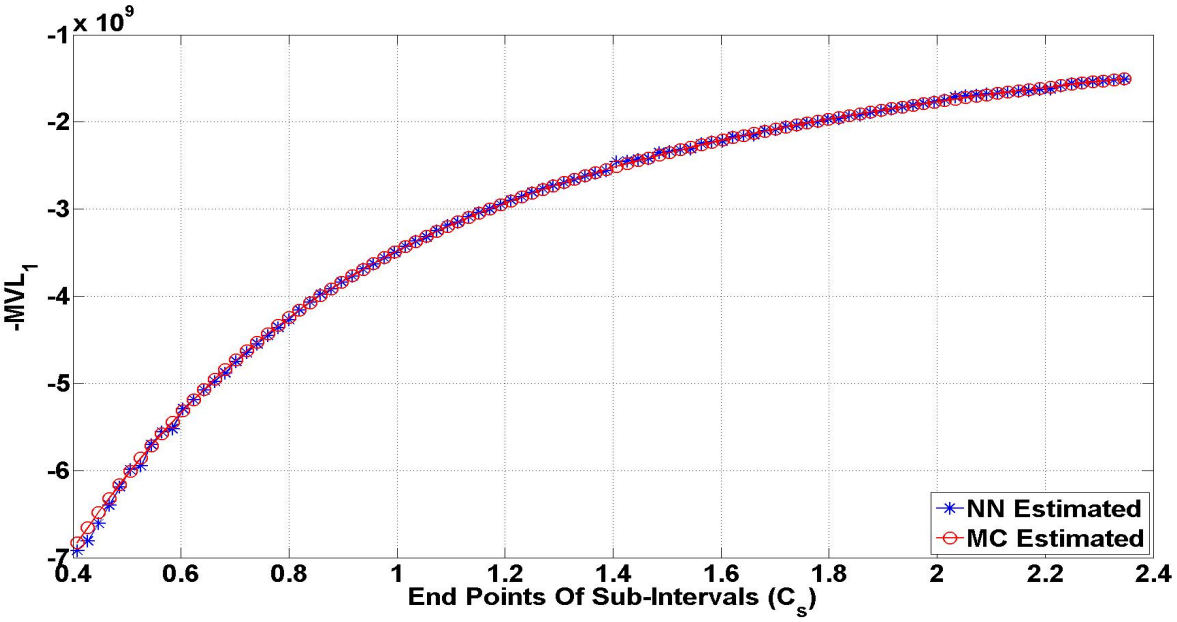
Value Of Interest	Relative Error (%)					
	S1	S2	S3	S4	S5	S6
SCR	-0.85	0.69	-0.81	-3.58	3.02	1.52
MVL_0	0.22	0.52	0.96	-0.36	-0.27	0.70
$\text{MVL}_1^{(99.5)}$	0.43	0.11	0.91	0.97	-1.20	-0.05

Table 4.4: Relative error in the estimation of the current liability value, one year liability value, and the SCR for the input portfolio.

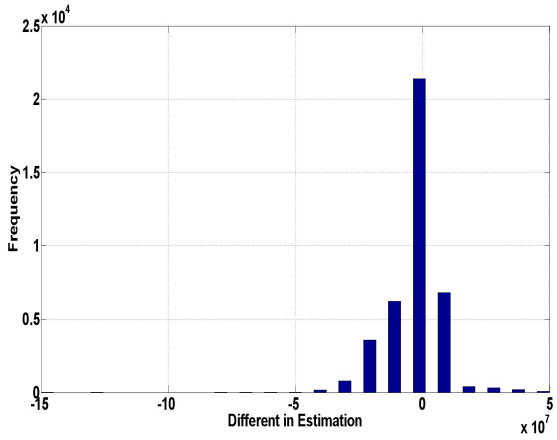
to use piecewise linear interpolation to estimate the value of the liability for points inside the sub-intervals. However, the estimation of this curve using the proposed neural network framework does not result in as smooth a curve as we had hoped. Therefore, we might be able to increase the accuracy of the proposed framework by using a non-linear curve fitting technique. Notice that, because of the particular simple asset structure that we use in this chapter the difference between the liability values in one year's time and the corresponding Δ values is a constant.

As we mention in Section 4.2 and earlier in this section, in this chapter, we do not address the issue of the choice of the interpolation scheme to estimate the Δ values of the $\text{P}^{(i)}, 1 \leq i \leq 40,000$, paths. Because of that and to have a fair pointwise comparison between the proposed neural network technique and the MC technique, we avoid using different interpolation schemes to estimate liabilities for the $\text{P}^{(i)}, 1 \leq i \leq 40,000$, paths.

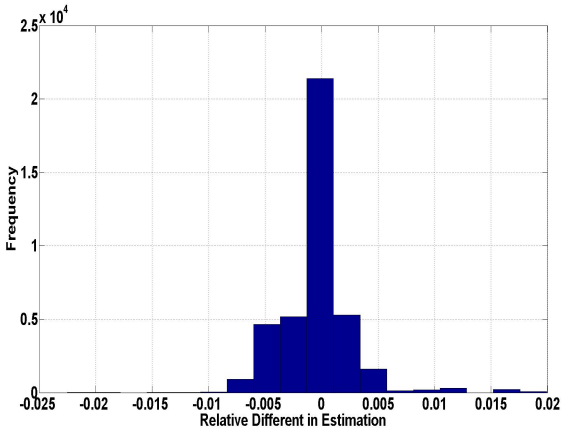
Table 4.5 presents the statistics on the running time of the proposed neural network approach for the nested simulation framework (denoted as NN in this table and elsewhere in the chapter) and the nested MC simulation framework. The results suggest a speed-up of $4-8$ times, depending on the scenario, and an average speed-up of 6 times. Considering that the implementation of the neural network was sequential and we compared the running time of the neural network with the implementation of MC simulations that uses parallel processing on 4 cores, we observe that even a simple implementation of the



(a) Estimated one-year liability values computed by the neural network framework and the MC method.



(b) Histogram of the difference at each end point of sub-intervals in estimation of the liability via the neural network approach and the MC simulations.



(c) Histogram of the relative difference (4.7) at each end point of sub-intervals in estimation of liability via the neural network approach and the MC simulations.

Figure 4.3: Comparing estimation of one-year liability values of the input portfolio computed by the proposed neural network framework and the MC method.

Method	Running Time	
	Mean	STD
MC	49334	0
NN	8370	2465

Table 4.5: simulation time of each method to estimate the SCR. All times are in seconds.

neural network can be highly efficient. As noted earlier, there is significant potential for parallelism in our neural network approach as well. Exploiting this parallelism should further improve the running time of the neural network. We plan to investigate this in the future. In addition, notice that we used a moderate number of MC simulation scenarios compared to the suggested values in [4]. As we mention earlier, an increase in the number of MC scenarios will not increase the running time of our neural network significantly, because we only need MC simulations for the representative contracts and for the validation/training portfolio; however, it increases the running time of the MC simulations significantly.

Chapter 5

Sampling Method

In our experiments so far in this thesis, for simplicity of analysis, we have only studied uniformly distributed synthetic portfolios of VA contracts. However, insurance companies often deal with VA portfolios that are non-uniformly distributed in the space in which the portfolio is defined. As we discuss later in the chapter, our numerical experiments on a non-uniform input portfolio shows that our proposed neural network framework, when used with the uniform sampling of Chapter 3 to estimate the delta value of the portfolio, is less accurate than when it is used to estimate the delta value of a uniform input portfolio. Therefore, we need to change the sampling method to improve the accuracy of our neural network approach.

In this chapter, we propose a non-uniform sampling method that provides samples that have a probability distribution that is similar to that of the input portfolio. Our experimental results show that the proposed sampling method, if used to create the representative portfolio and the training portfolio, achieves better accuracy results – lower mean and standard deviation of recorded relative errors.

5.1 Design of the Sampling Method

A non-uniform portfolio fills the space in which it is defined in a way that creates concentration of points in some parts of the space and sparse regions in other parts of the space. A sample portfolio that has more representative policies in parts of the space that is densely filled with VAs of the input portfolio allows us to have a finer resolution of risk values in that part of the space and hence can potentially increase the accuracy of our estimation. If we generate the sample portfolio according to the probability distribution of the input portfolio (i.e., the distribution of the contracts in the input portfolio in the space in which the input portfolio is defined), we can exploit this property to obtain better accuracy.

To generate the sample portfolio according to the probability distribution of the input portfolio, we need to approximate the true probability distribution of input portfolio first. An accurate estimation of the true probability distribution should also consider the statistical dependency of per attribute distributions. But such probability estimators are complex and using them to estimate the probability distribution of the input portfolio can significantly increase the running time of the spatial interpolation framework. Furthermore, the ultimate goal of our spatial interpolation framework is not to accurately estimate the true probability distribution of the input portfolio, but rather to estimate accurately and efficiently the risk metrics associated with the portfolio. Therefore to manage the computational complexity of the proposed neural network framework, in what follows, we propose a sampling method that assumes statistical independence of attributes that define the space in which the input portfolio is defined. Our numerical results (refer to Section 5.2) show that even with correlated probability distributions of the input portfolio, the proposed sampling method allows the proposed neural network approach to provide fairly accurate estimations of the Greeks for large portfolios of VA contracts.

The probability distribution for numerical attributes of VAs, such as account value

and guarantee value, are continuous variables. In practice, we need to discretize the domain of these attributes to be able to find a probability distribution (an approximation of the true probability distribution) for them. The number of discretization points determines the accuracy of the estimated probability distribution. It also determines the cost of computing the probability distribution and taking samples from that probability distribution. The computational cost and accuracy are both increasing functions of the number of discretization points. Hence, we aim to find the right balance between accuracy and computational complexity when we choose the number of discretization points. Similarly, for numeric attributes that have a discrete domain, we may choose a subset of the points in the domain of these numeric attributes to reduce the computational complexity while providing accurate estimations of the true probability distribution. To keep our sampling method simple, we propose to divide the domain of numeric attributes into sub-intervals of almost equal length and take the end points of these sub-intervals as our discretization points that we use to generate the representative portfolio. As we discuss in Chapter 3, to avoid over-fitting the data, the end points of the training portfolio and the representative portfolio cannot be too similar. Therefore, we propose to divide each of the above-mentioned sub-intervals into almost equal length sub-sub-intervals and take only the end points of those sub-sub-intervals that are internal points of the sub-intervals as our discretization points that we use to generate the training portfolio.

Let $c_1, c_2, \cdots, c_n$ represent a set of end points defined for attribute A. Moreover, assume m and M are, respectively, the minimum and the maximum values of the attribute A in the input portfolio. For each $i \in \{1, 2, \cdots, n\}$, we associate an interval $[a_i, b_i]$ with end point c_i. For $i > 1$, we set $a_i = \frac{c_i + c_{i-1}}{2}$ and we let $a_1 = \min(c_1, m)$. If $i < n$, we set $b_i = \frac{c_i + c_{i+1}}{2}$ and we let $b_n = \max(c_n, M)$. Now, we can use $[a_i, b_i], 1 \leq i \leq n$, to assign empirical probability values to $c_i, 1 \leq i \leq n$.

To increase the accuracy of the estimation, we estimate the probability distribution of each attribute for each type of VA that exists in the input portfolio separately. For

example, if the input portfolio consists of GMWB and GMAB riders, we compute two probability distributions, one for the VAs with the GMWB rider and one for VAs with the GMAB riders. However, for attributes that are similar between the the two riders, we use the same end points.

As noted at the beginning of this section, we are interested in measuring the spatial distribution of VAs. In particular, we want to measure the density by which VAs fill each part of the space. Therefore, for an attribute A, with end points $c_i, 1 \leq i \leq n$, and associated intervals $[a_i, b_i], 1 \leq i \leq n$, we compute the empirical probability distribution of A for the VAs with rider r as follows:

$$p(c_i) = F^{-1} \times \frac{N_i^{(r)}}{(b_i - a_i)},\ 1 \leq i \leq n \tag{5.1}$$

where $N_i^{(r)}$ defines the number of VAs in the input portfolio with rider r for which the value of attribute A is in $[a_i, b_i]$ and F^{-1} is the normalization factor. The value $\frac{N_i^{(r)}}{b_i - a_i}$ measures the spatial density of points in the interval $[a_i, b_i]$ and hence $p(c_i)$, in essence, approximates the probability density function of values in the interval $[a_i, b_i]$. For two intervals $[a_i, b_i]$ and $[a_j, b_j]$ of different size (i.e., $b_i - a_i \neq b_j - a_j$), equation (5.1) allows us to assign both intervals equal probabilities so long as the density of points in the two intervals are equal– i.e., $\frac{N_i^{(r)}}{(b_i - a_i)} = \frac{N_j^{(r)}}{(b_j - a_j)}$. This is particularly important for the end points c_1 and c_n for which the size of associated sub-intervals $[a_1, b_1]$ and $[a_n, b_n]$ might be smaller than the size of other sub-intervals. For example, for a uniformly distributed input portfolio, we want the end points $c_i, 1 \leq i \leq n$, to all have equal chance of occurrence in the sample. In other words, we want $p(c_i) = p(c_j), 1 \leq i, j \leq n$. Now assume for this input portfolio, we have chosen $c_i, 1 \leq i \leq n$, values such that $a_1 = c_1$ and $b_n = c_n$. Because there exists no values smaller than $c_1 = a_1$ or bigger than $c_n = b_n$ and the sub-intervals $[c_i, c_{i+1}], 1 \leq i \leq n-1$, are of almost equal length, the size of the intervals $[a_1, b_1]$ and $[a_n, b_n]$ are half the size of the internal intervals $[a_i, b_i], 1 < i < n$. Therefore, $N_1^{(r)} \approx \frac{1}{2} N_i^{(r)}, 1 < i < n$, $N_n^{(r)} \approx \frac{1}{2} N_i^{(r)}, 1 < i < n$, and

$N_i^{(r)} \approx N_j^{(r)}, 1 < i, j < n$. Thus, the $b_i - a_i$ term in the denominator of (5.1) is necessary to ensure that the $p(c_i), 1 \leq i \leq n$, values are approximately equal and hence to provide that the empirical probability distribution approximates the uniform distribution.

Calculating the probability distribution of categorical attributes is much easier. A categorical attribute has a discrete probability distribution with typically a few categories only. For the VAs considered in this thesis, the number of categories do not exceed 3. Hence, we can easily calculate the empirical probability distribution for all of the categories. For an attribute A with categories $c_1, c_2, \cdots, c_n$, we compute the empirical probabilities of VAs with rider r as

$$p(c_i) = \frac{N_i^{(r)}}{N_r},\ 1 \leq i \leq n \tag{5.2}$$

where $N_i^{(r)}$ is the number of VAs in the input portfolio with rider r for which the category of attribute A is c_i and N_r is the number of VAs in the input portfolio with rider r.

After computing the empirical probability distributions of attributes for a rider r by (5.1) or (5.2), we can generate a sample VA portfolio with rider r by selecting the attributes of the sample VAs independently of each other and according to the computed probability distributions. One major problem with this method of generating samples is that it is likely that we generate duplicate sample VAs, especially if the sample size is big and/or the probability distribution of the attributes is very skewed. As we discuss in Chapter 3, duplicate VAs can decrease the accuracy of the proposed neural network by over emphasizing the influence that the value of duplicate sample VAs have on the value of VA policies in the space. Therefore, if we generate a duplicate VA, we discard it and generate another VA sample. Notice that, if we just discard a duplicate VA without generating another, the size of the sample varies from one experiment to the next. As we demonstrate in Section 3.2.6, the size of the sample portfolios affects the accuracy and the running time of the neural network framework. Therefore, to minimize these effects, we choose to replace the duplicate VAs with other sample VAs. In other words,

we generate sample VAs, discarding duplicates, until we have a sample portfolio with a pre-defined size.

One major problem with discarding duplicate VAs and replacing them with other sample VAs is that the realized probability distribution of the generated sample portfolio may no longer be close to the computed empirical probability distribution. To alleviate this problem in a simple manner, we suggest an iterative approach to generating the sample portfolio. In each iteration, we generate a sample VA portfolio and count the number of duplicate VAs that we discarded in the process. Among the generated sample VA portfolios, we choose the one with the least number of duplicates. There might be more than one sample VA portfolio with the least number of duplicate VAs. To break the ties, we choose the sample with the minimum MSE of the relative error between the computed empirical probability distribution for the input portfolio and the sample VA portfolio. The aforementioned MSE measures the closeness of realized probability distribution of the sample portfolio to the computed empirical probability distribution of the input portfolio.

To compute the above-mentioned MSE value, we first calculate the per rider per attribute empirical probability distribution of each sample VA portfolio. The calculation is similar to the method described above for computing the empirical probability distribution of the input portfolio–i.e., values a_i, b_i and $c_i, 1 \leq i \leq n$, defined above and equations (5.1) and (5.2). We then compute the MSE of the relative error between the computed empirical probability distribution for the input portfolio and the sample VA portfolio as follows

$$MSE = \sqrt{\frac{1}{\sum_{r \in R} \sum_{a \in A_r} n_a} \sum_{r \in R} \sum_{a \in A_r} \sum_{i=1}^{n_a} \left(\frac{P_S(c_i; a, r) - P_I(c_i; a, r)}{P_I(c_i; a, r)} \right)^2} \tag{5.3}$$

where R is the set of riders in the input portfolio, A_r is the set of attributes for rider r, n_a is the number of end points associated with attribute a, c_i is the i^{th} end point, $P_I(\cdot; r, a)$ is the empirical probability distribution of attribute a of rider r in the input

portfolio and $P_S(\cdot; r, a)$ is the empirical probability distribution of attribute a of rider r in the sample portfolio.

Once we know the size of the sample portfolio that we want to generate, we need to decide the number of samples that we want to have for each rider. Intuitively, one might generate samples from each VA rider according to the frequency of occurrence of that rider in the input portfolio. For example, for the synthetic input portfolio of 100,000 VAs that we use in Section 3.2, we should choose approximately 50% of the samples to have the GMDB rider while the rest of the samples to have the GMDB + GMWB rider.

We used the neural network of Section 3.2 with the sampling method proposed in this section to estimate the delta of the synthetic portfolio of 100,000 VAs described in Section 3.2. We use the proposed sampling method to generate the representative portfolio and the training portfolio. In the generated sample portfolios, the number of VAs having the GMDB rider is almost equal to the number of VAs having the GMDB + GMWB rider. We used the same per attribute end points as we did in our experiments of Section 3.2 (see Tables 3.2 and 3.3).

A MC simulation approach is considered to provide accurate estimations of a key risk metric if the standard deviation in outcomes of various runs of the MC simulations is far less than 1% of the mean value of these outcomes. In pursuit of estimation methods that are more efficient than an accurate MC simulation approach, insurance companies consider only estimation methods for which the absolute value of the relative error compared to the outcome of a run of the MC simulations is at most 5%. As we discuss in Chapters 2 and 3, MC simulations with $10,000$ inner loop scenarios provide accurate estimations of the portfolio delta value for the portfolio of VAs that we consider in this set of experiments. Therefore, the neural network method of Section 3.2 is a good alternative to these MC simulations if each run of this method has an absolute relative error of less than 5%.

We did 5 independent experiments each with a different set of the representative port-

folio, the training portfolio and the validation portfolio. The absolute value of the relative error of estimating the delta for the input portfolio in most of the above-mentioned experiments was bigger than 5%. In each run, the generated sample portfolios had a per rider empirical distribution that was very close to uniform with fewer than 3 duplicate VAs in their generation process. Hence, generating samples that are not uniformly distributed in the space is not the cause of the poor accuracy.

Comparing the sampling method proposed in this section with the uniform sampling method of Section 3.2 sheds light on an important issue. Comparing the attributes that we use to characterize the VAs of the synthetic input portfolio of Section 3.2, we see that the two groups of VAs in the input portfolio differ in only one attribute, the withdrawal rate. The VAs with the GMDB rider do not need a withdrawal rate. However, the VAs with the GMDB + GMWB rider need a withdrawal rate. Consequently, the space in which the VAs with GMDB + GMWB riders are defined has an extra dimension compared to the space in which the GMDB riders are defined. Hence, intuitively, we need more samples of VAs with GMDB + GMWB riders, compared to VAs with GMDB riders, to provide the information required for the added dimension. In fact, this is what the uniform sampling method of Section 3.2 does. The uniform sampling method for each of the representative portfolio and the training portfolio first creates a portfolio consisting of all the combinations of end points in Tables 3.2 and 3.3 respectively. In those sample portfolios, because of the number of withdrawal rates that are used as end points, the number of VAs with GMDB + GMWB riders are twice and thrice, respectively, the number of VAs with GMDB riders. GMDB riders, in this comparison, can be considered as having a withdrawal rate of zero. The uniform sampling method then eliminates VAs uniformly at random from the sample portfolios. Because the elimination is uniformly at random, the number of eliminated sample VAs from each rider is almost proportional to the number of VAs that was originally created for that rider. Hence, after the elimination, the number of VAs with GMDB + GMWB riders are still almost twice/thrice of the

number of VAs with GMDB riders.

This observation suggests that we should measure the importance of each attribute for each rider by the number of points that we need for that attribute to accurately estimate the risk metric of interest. In other words, the number of VAs for rider r should be proportional to $I_r = \prod_{a \in A_r} n_a^{(r)}$, where A_r is the set of attributes that characterizes rider r and $n_a^{(r)}$ is the number of end points defined for attribute a of rider r. The product I_r determines the number of vertices on the grid in the space of rider r that is necessary for accurate estimation of the risk metric of interest for rider r.

A grid with I_r vertices is necessary only if the VAs in the input portfolio sufficiently fill the space of rider r. In other words, the space of rider r is not sparsely filled. Even then, the accuracy in estimation of the risk metric of interest for VAs with rider r can affect the overall (portfolio level) accuracy of our estimation to the extent that VAs with rider r are present in the input portfolio. If VAs with rider r make up only a small fraction of the input portfolio and/or rider r does not significantly affect the risk metric of interest, we may not benefit from a highly accurate estimation of rider r when computing the risk metric of interest. In our neural network framework, the sampling happens before we begin estimating the values. Therefore, we are not able to quantify the significance of each rider on the risk metric of interest to do the sampling. However, we can quantify the percentage by which each rider fills the input portfolio. Therefore, we propose to choose the number of VAs with a particular rider r in the sample portfolio to be proportional to the number of VAs with that rider in the input portfolio (N_r).

Based on the discussion above, the number of VAs in the sample portfolio should be proportional to both I_r and N_r. Therefore, we propose to split the VAs of the sample portfolio amongst different riders according to the following distribution

$$f_r = \frac{N_r \times \prod_{a \in A_r} n_a^{(r)}}{\sum_{r' \in R} \left(N_{r'} \times \prod_{a \in A_{r'}} n_a^{(r)} \right)} \tag{5.4}$$

where R is the set of riders in the input portfolio, N_r is the number of VAs in the input

portfolio with rider r, A_r is the set of attributes that characterizes rider r, $n_a^{(r)}$ is the number of end points defined for attribute a and f_r is the fraction of samples that should be dedicated to rider r. The denominator of equation (5.4) is for normalization.

We regard the f_r in equation (5.4) as elements of a probability distribution. In other words, in the proposed sampling method, we, first, assign each sample VA a rider according to the probability distribution associated with (5.4) and then proceed to find the attributes of the sample VA according to equations (5.1) and (5.2). To summarize, the sampling method works as follows.

Algorithm 1 Proposed Sampling Method

Input: Input portfolio, Size of the sample portfolio (n_s), and maximum number of iterations ($\text{iter}_{\max}$)

Select end points of sub-intervals for each attribute of each rider
Compute the empirical probability dist. of numerical attributes by (5.1)
Compute the empirical probability dist. of categorical attributes by (5.2)
Compute the probability dist. of riders by (5.4)
best-sample ← generate a sample of size n_s according to the distributions associated with (5.4), (5.1), and (5.2)
best-dup ← number of duplicates generated in creating the sample
best-mse ← compute the mse of the generated sample by (5.3)
for $i = 1$ to $\text{iter}_{\max} - 1$ **do**
    sample ← generate a sample of size n_s according to the distributions associated with (5.4), (5.1), and (5.2)
    dup ← number of duplicates generated in creating the sample
    mse ← compute the mse of the generated sample by (5.3)
    if (dup < best-dup) **then**
        best-sample ← sample
        best-dup ← dup
        best-mse ← mse
    else if (dup = best-dup **AND** mse < best-mse) **then**
        best-sample ← sample
        best-dup ← dup
        best-mse ← mse
    end if
end for
return best-sample

In the next section, we demonstrate the accuracy and time efficiency of Algorithm 1.

5.2 Numerical Experiments

In this section, we investigate the performance of the proposed sampling method by using it in the neural network framework described in Chapter 3 to find the delta value of synthetic portfolios of VAs. We use the sampling method proposed in the previous section to find the representative portfolio and the training portfolio in each experiment. As in Section 3.2, the validation portfolio consists of 250 VA contracts chosen uniformly at random from the input portfolio. Our experiments, implemented in Java, are run on a machine with dual quad-core Intel X5355 CPUs.

5.2.1 Uniform Input Portfolio

In the first set of experiments, we use the proposed sampling method to find the delta value of the synthetic portfolio of 100,000 VAs that we describe and use in Section 3.2. We already know that, when the proposed neural network framework uses the uniform sampling method of Chapter 3, it can provide accurate results. Hence, these experiments test if the proposed neural network remains accurate when using the sampling method proposed in this chapter.

As we mention earlier, the sampling method proposed in this chapter assumes statistical independence of attributes that define the space in which the input portfolio is defined. Therefore, this sampling method should work best with input portfolios for which the per attribute marginal distributions are independent. To measure the statistical dependence between these marginal distributions, we choose to use their correlation coefficient and their Randomized Dependence Coefficient (RDC) [48]. The RDC is a measure, defined in $[0, 1]$, of non-linear dependence between random variables that is

easy to implement and has low computational cost[1]. Furthermore, it is invariant with respect to monotonically increasing transformations of the marginal distributions and estimates Hirschfeld-Gebelein-Renyis Maximum Correlation Coefficient, which is a measure that satisfies all the fundamental properties of a measure of dependence described by Renyi [60]. Compared to commonly used and theoretically known measures such as Person's rho, Spearman's Rank or Kendall's tau, which only consider association patterns like linear or monotonically increasing functions, RDC considers a broader class of non-linear dependencies [48].

Tables 5.1 and 5.2 contain, respectively, the pairwise correlation coefficient and RDC of attributes for the input portfolio. Except for the entries corresponding to the value of the correlation coefficient and the RDC for the rider and the withdrawal rate attributes, all the other entries in Table 5.2 are very small, which suggests that the attributes of the input portfolio are statistically independent. The correlation coefficient and the RDC value corresponding to the rider and withdrawal rate attributes is big only because we have chosen the withdrawal rate to be zero for GMDB riders. Therefore, it is fair to assume that the attributes of the input portfolio are statistically independent and hence expect the proposed neural network framework when using the proposed sampling method to provide accurate estimations.

The setup of the neural network is similar to Section 3.2. The choice of the end points of each attribute for the representative portfolio and the training portfolio is defined in Tables 3.2 and 3.3, respectively. Notice that in both tables the choice of the end points determines a set of almost equal length sub-intervals for each attribute. Moreover, the attributes for both types of riders are the same except for the withdrawal rate[2]. In other words, the only difference between the attributes of the two riders is that the

[1]The implementation of RDC requires an appropriate choice of free parameters k, which determines the number of non-linear transformations, and s, which determines the randomness of linear projections [48]. In our experiments, following the guidelines of [48], we use $k = 10$ and $s = 1/10$, which provides reliable and stable results.

[2]The GMDB riders are assumed to have a withdrawal rate of zero.

Attributes	Corr. Coef.	Attributes	Corr. Coef.	Attributes	Corr. Coef.
(Rider, Gender)	0.0011	(Rider, Age)	-0.0017	(Rider, AV)	0.0000
(Rider, GV)	0.0012	(Rider, WR)	0.9492	(Rider, Mat)	0.0011
(Gender, Age)	0.0044	(Gender, AV)	-0.0027	(Gender, GV)	0.0014
(Gender, WR)	-0.0002	(Gender, Mat)	0.0064	(Age, AV)	0.0005
(Age, GV)	-0.0015	(Age, WR)	-0.0011	(Age, Mat)	-0.0013
(AV, GV)	0.0027	(AV, WR)	-0.0013	(AV, Mat)	0.0009
(GV, WR)	0.0013	(GV, Mat)	0.0013	(WR, Mat)	0.0019

Table 5.1: Correlation coefficient between pairs of attributes in the synthetic uniform input portfolio defined in Section 3.2.

GMDB+GMWB rider has the withdrawal rate as an extra attribute. This choice of attributes affects equation (5.4), which simplifies to

$$f_r = \frac{N_r \times n_{wr}^{(r)}}{N_{\text{GMDB}} \times n_{wr}^{(\text{GMDB})} + N_{\text{GMWB+GMDB}} \times n_{wr}^{(\text{GMDB+GMWB})}} \tag{5.5}$$

where $r \in \{\text{GMDB, GMDB+GMWB}\}$, $n_{wr}^{(\text{GMDB})} = 1$ and for the representative portfolio $n_{wr}^{(\text{GMDB+GMWB})} = 2$ (refer to Table 3.2) and for the training portfolio $n_{wr}^{(\text{GMDB+GMWB})} = 3$ (refer to Table 3.3). We use a maximum number of iterations of $20,000$ in Algorithm 1 to generate both the representative portfolio and the training portfolio.

We record the relative error of the estimated delta values using equation (3.19) as the measure of accuracy of our framework. In Table 5.3, the performance of the neural network framework when employed with uniform sampling of Chapter 3 is compared to the performance of the neural network framework when used with the sampling method proposed in Section 5.1. To be fair in our comparisons, the data in Table 5.3 for the proposed sampling method was gathered from 6 independent experiments in a manner similar to that used to gather the data for the uniform sampling method. The data in

Attributes	RDC	Attributes	RDC	Attributes	RDC
(Rider, Gender)	0.0011	(Rider, Age)	0.0073	(Rider, AV)	0.0041
(Rider, GV)	0.0067	(Rider, WR)	1.0000	(Rider, Mat)	0.0072
(Gender, Age)	0.0083	(Gender, AV)	0.0079	(Gender, GV)	0.0037
(Gender, WR)	0.0075	(Gender, Mat)	0.0083	(Age, AV)	0.0123
(Age, GV)	0.0100	(Age, WR)	0.0156	(Age, Mat)	0.0103
(AV, GV)	0.0149	(AV, WR)	0.0072	(AV, Mat)	0.0124
(GV, WR)	0.0096	(GV, Mat)	0.0094	(WR, Mat)	0.0098

Table 5.2: Randomized dependence coefficient (RDC) with $k = 10$ and $s = 1/10$ pairs of attributes in the synthetic uniform input portfolio defined in Section 3.2.

Sampling Method	Relative Error (%)		Running Time	
	Mean	STD	Mean	STD
Uniform Sampling	0.38	1.35	539	120
Proposed Sampling	−0.50	1.88	622	113

Table 5.3: Statistics on the running time and accuracy of the neural network framework when used with the uniform sampling method of Chapter 3 and the sampling method proposed in Section 5.1 to estimate the delta value of a uniformly distributed input portfolio. The recorded errors are relative errors as defined in (3.19). All times are in seconds.

Table 5.3 shows that the neural network framework with the proposed sampling method is accurate – low mean and standard deviation. However, the standard deviation is larger than for the uniform sampling method. The number of iterations are large enough that, in almost all of the experiments, we can find sample portfolios for which the number of generated duplicate VAs is zero. Therefore, the increase in the standard deviation of the relative error is not caused by the generation of duplicate VAs. However, our

proposed sampling method, unlike the uniform sampling method, has to estimate the empirical probability distribution of the input portfolio. Thus, one possible explanation for the increase in the standard deviation is that the error in estimation of the empirical probability distribution results in generation of sample portfolios that do not uniformly and effectively fill the space in which the input portfolio is defined, resulting in poorer estimations. In Section 5.2.3, we provide more evidence that supports this conjecture.

Because of the extra overhead in computing the empirical probability distribution and more importantly the iterative process involved in generation of the sample portfolios, the sampling method proposed in Section 5.1 has a larger running time than the uniform sampling method of Section 3.2.

5.2.2 Non-Uniform Input Portfolio With Low Correlation

Proposing a sampling method that is more complicated than the uniform sampling method of Section 3.2 only makes sense if it provides a better performance on non-uniform input portfolios. To demonstrate the need for a sampling method that is more sophisticated than the uniform sampling method, in the following experiments, we use the neural network framework of Chapter 3 to estimate the delta value of synthetic portfolios of 100,000 VAs that are non-uniformly distributed in the space. To investigate only the effect of the non-uniformity of the distribution, we first use an input portfolio of VAs that is non-uniformly distributed, but the attributes of VA contracts in this portfolio are statistically independent. Because the attributes used to describe the input portfolio of this section and the one used in experiments of Section 5.2.1 are similar and are statistically independent, we can compare the performance results of the experiments of this Section with the performance results of Section 5.2.1 to gain a sense of the effect that the non-uniformity of the distribution has. In Section 5.2.4, we consider a non-uniformly distributed input portfolio that has statistically dependent attributes. This allows us to gain a sense of the effect that statistical dependency has on the performance of the

Attribute	Distribution
Guarantee Type	30% GMDB and 70% GMDB + GMWB
Gender	40% Male and 60% Female
Age	Normal Distribution with Mean 40 and Standard Deviation 7
Account Value	Uniformly at Random From $[1e4, 5e5]$
Guarantee Value	Uniformly at Random From $[0.5e4, 6e5]$
Withdrawal Rate	Uniformly at Random From $\{0.04, 0.05, 0.06, 0.07, 0.08\}$
Maturity	Normal Distribution with Mean 17 and Standard Deviation 3

Table 5.4: Distribution of attributes in the input portfolio.

proposed sampling method.

In this experiment, the input portfolio is non-uniformly distributed in the space defined by the attributes of Table 3.1. Table 5.4 describes the distribution of each attribute in the space. In Table 5.4, although the age and the maturity values are referred to as normally distributed, they are integer values rounded from a set of real values with the corresponding normal distribution and restricted to the intervals $[20, 60]$ and $[10, 25]$, respectively.

The data in Tables 5.5 and 5.6 show, respectively, the correlation coefficient and the RDC value for each pair of attributes. Similar to the data in Tables 5.1 and 5.2, the entries in these tables are small enough, except for the correlation coefficient/RDC value of rider and withdrawal rate, that we can assume the attributes of the input portfolio are statistically independent. The correlation coefficient/RDC value corresponding to the rider and withdrawal rate attributes is big because we choose to assign a value of zero to withdrawal rate of VA accounts with the GMDB rider. Although, the percentage of VA accounts in the input portfolio with GMDB rider is smaller than that of the input portfolio of Section 5.2.1, the percentage of GMDB riders is still significant enough to cause the correlation coefficient to be big and the RDC score to be one.

Attributes	Corr. Coef.	Attributes	Corr. Coef.	Attributes	Corr. Coef.
(Rider, Gender)	0.0017	(Rider, Age)	0.0005	(Rider, AV)	-0.0026
(Rider, GV)	0.0017	(Rider, WR)	0.9184	(Rider, Mat)	-0.0036
(Gender, Age)	0.0060	(Gender, AV)	-0.0037	(Gender, GV)	-0.0032
(Gender, WR)	0.0017	(Gender, Mat)	-0.0024	(Age, AV)	0.0024
(Age, GV)	0.0005	(Age, WR)	0.0002	(Age, Mat)	0.0002
(AV, GV)	-0.0015	(AV, WR)	0.0005	(AV, Mat)	-0.0036
(GV, WR)	0.0005	(GV, Mat)	0.0016	(WR, Mat)	-0.0046

Table 5.5: Correlation coefficient between pair of attributes in the synthetic non-uniform input portfolio defined in the space of Table 5.4.

Attributes	RDC	Attributes	RDC	Attributes	RDC
(Rider, Gender)	0.0017	(Rider, Age)	0.0075	(Rider, AV)	0.0079
(Rider, GV)	0.0067	(Rider, WR)	1.0000	(Rider, Mat)	0.0046
(Gender, Age)	0.0086	(Gender, AV)	0.0085	(Gender, GV)	0.0043
(Gender, WR)	0.0030	(Gender, Mat)	0.0076	(Age, AV)	0.0097
(Age, GV)	0.0125	(Age, WR)	0.0107	(Age, Mat)	0.0116
(AV, GV)	0.0134	(AV, WR)	0.0136	(AV, Mat)	0.0105
(GV, WR)	0.0114	(GV, Mat)	0.0112	(WR, Mat)	0.0122

Table 5.6: Randomized dependence coefficient (RDC) with $k = 10$ and $s = 1/10$ pair of attributes in the synthetic non-uniform input portfolio defined in the space of Table 5.4.

The parameters of the neural network are similar to those used in Section 3.2. However, we choose a degree 4 polynomial to fit the smoothed MSE values when using the sampling method proposed in Section 5.1. The proposed sampling method (i.e., Algorithm 1) uses 40,000 iterations to provide the representative contracts and 20,000 iterations to provide the training portfolio. Table 5.7 lists the performance statistics

Sampling Method	Relative Error (%)		Running Time	
	Mean	STD	Mean	STD
Uniform Sampling	−1.21	2.24	623	230
Proposed Sampling	−0.84	1.46	655	213

Table 5.7: Statistics on the running time and accuracy of the neural network framework when used with the uniform sampling method and the sampling method proposed in Section 5.1 to estimate the delta value of a non-uniformly distributed input portfolio. The recorded errors are relative errors as defined in (3.19). All times are in seconds.

for the neural network framework of Chapter 3 using the uniform sampling method and the sampling method proposed in Section 5.1 to estimate the delta value of the input portfolio. The statistics are derived from 6 independent runs. In each run a new set of representative portfolio, training portfolio, and validation portfolio is used.

The mean and standard deviation of the relative error for the uniform sampling method in Table 5.7 is significantly larger than for the uniform input portfolio in the previous section. In other words, the accuracy has decreased significantly. In particular, the high standard deviation value implies that the method is less stable and hence less reliable for the non-uniform portfolio of this section than for the uniform portfolio of the previous section. The statistics for the sampling method proposed in Section 5.1 are much better than the corresponding statistics for the uniform sampling method. This supports our belief that the proposed sampling method is more accurate than the uniform sampling method when applied to non-uniform portfolios.

Another interesting observation is that the running times of the neural network framework under both sampling methods are almost equal. One interpretation of this observation is that the bad samples provided by the uniform sampling method has increased the running time of the method. We substantiate this explanation in the next section.

In generation of the representative portfolios for the non-uniform input portfolio of

this section, our sampling method proposed in Section 5.1 on average generates 9 duplicate VA samples. Considering that we have doubled the number of iterations from $20,000$ to $40,000$ iterations, compared to the experiments of the previous section, we can see that increasing the number of iterations any further is not an effective way to reduce the number of duplicate VA samples in hope of achieving better accuracy. Therefore, we should accept that having duplicates cannot be avoided and look for other strategies to improve the accuracy of the proposed sampling method. It is worth mentioning that, in our experiments, we were always able to generate the training portfolio from a sample portfolio for which no duplicate VA samples were generated.

5.2.3 Sobol Sequence

We note in Sections 3.1.4 and 5.1 that a good sample for the proposed neural network framework should avoid duplicates and should consist of well-spaced sample VAs. Because of that, in our proposed sampling method, we discard duplicates knowing that discarding the duplicates might create a sample with an empirical distribution that may not be similar to the empirical distribution of the input portfolio.

Java, like most programming languages, has a default random number generator that uses a linear congruential formula to generate random numbers [1]. Linear congruential random number generators do not have low discrepancy[3]. In other words, the sequence of numbers generated by these pseudo-random number generator does not uniformly fill the space. Hence, it is highly likely that they create duplicate values if used in the sampling method proposed in Section 5.1 to generate the attributes of sample VAs.

Several quasi-random number generators are proposed in literature [10] to generate low-discrepancy sequences. One of the simplest to implement and most commonly used is Sobol's quasi-random number generator [64]. In the next set of experiments, we use

[3]Discrepancy is a measure of good spacing. The discrepancy of a sequence is low if the numbers of values in the sequence that fall into an arbitrary set A is almost proportional to the measure of A.

Sampling Method	Relative Error (%)		Running Time	
	Mean	STD	Mean	STD
Proposed Sampling With Sobol Rand	−0.49	1.11	556	64

Table 5.8: Statistics on the running time and accuracy of the neural network framework when used with the sampling method proposed in Section 5.1 and the Sobol quasi-random number generator to estimate the delta value of a non-uniformly distributed input portfolio. The recorded errors are relative errors as defined in (3.19). All times are in seconds.

an efficient implementation of Sobol's quasi-random number generator [10] within the sampling method proposed in Section 5.1 to create attributes of our VAs. Our hope is that by using Sobol sequences, we can generate VAs that are well-spaced and have fewer duplicates.

A Sobol quasi-random number generator requires a set of direction numbers to generate random outputs. In our implementation, we use the set of direction numbers proposed in [44]. These direction numbers create random outputs that are not only well-spaced in the high-dimensional space in which they are generated, but also their projections into a lower-dimensional space are also well-spaced.

Table 5.8 contains the performance statistics of our proposed sampling method incorporating a Sobol random number generator to find the attributes of each VA. The neural network that we use for these experiments has the same set of parameters as the neural network for our last set of experiments with the proposed sampling method in Section 5.2.2. Moreover, the recorded statistics in Table 5.8 are the result of 6 independent runs for the method. Comparing the results of Tables 5.8 and 5.7, we can conclude that the proposed sampling method incorporating a Sobol quasi-random number generator provides more accurate estimations – i.e., the mean and standard deviation of the relative error is lower. Our proposed sampling method, even with incorporation of the Sobol

quasi-random number generator, on average generates 9 and 0 duplicate sample VAs in generation of the representative portfolio and the training portfolio, respectively. These numbers of duplicate VAs are similar to the ones that we documented in the previous section. However, with incorporation of the Sobol quasi-random number generator, the MSE scores of the generated sample portfolios for the representative portfolio and the training portfolios on average are reduced to 0.2243 and 0.2548, respectively, from the average MSE scores of 0.2556 and 0.2872 when Java's default congruential random number generator was used. The decrease in the MSE score shows that the portfolios generated by the incorporation of the Sobol quasi-random number generator have probability distributions that are closer to the estimated empirical probability distribution of the input portfolio and hence have more effectively filled the space of the input portfolio.

Another interesting point about the results of Table 5.8 is that, with incorporation of the Sobol quasi-random number generator, the running time has also decreased quite significantly compared to the running time in Table 5.7, which can be attributed to better choices of the representative portfolio and the training portfolio. These results provide further evidence supporting our claims in Sections 5.2.2 and 3.1.4 that good samples can reduce the training time.

5.2.4 Non-Uniform Input Portfolio With High Correlation

In previous experiments in this thesis, we consider only input portfolios that are defined by attributes that are statistically independent. The proposed sampling method also assumes statistical independence of attributes that define the input portfolio. However, in practice, we expect some degree of dependence to exist between attributes that define an input portfolio. Therefore, in this set of experiments, we use an input portfolio which is defined by attributes that have some degree of statistical dependence.

To determine statistical dependencies that are realistic, we use the data of two recent surveys [65,69] on portfolios of VA contracts to design the synthetic input portfolio. Both

of these surveys contain information from major insurance companies. We need to select the input portfolio on VA contracts with GMDB and GMDB + GMWB riders so that we can have a meaningful study of statistical dependency by comparing the results of the experiments of this section with the results of previous experiments. Thus, we choose the input portfolio to contain only GMDB and GMDB + GMWB riders.

In all the surveys that we studied, i.e., [43,65,68,69], we were unable to find statistics that help us decide what percentage of the input portfolio we should allocate to each type of VA products that we are interested in. Furthermore, in the data from surveys, there exists no evidence of statistical dependence between the rider of the contracts and other attributes that define these contracts. Therefore, as in Section 5.2.2, we choose to define the input portfolio such that 30% of the contracts have a GMDB rider and 70% of the contracts have a GMDB + GMWB rider, independently of how the attributes of these contracts are defined.

The GMDB rider was a popular VA contract in the 90s. However, in recent years, it has become an option that is offered along with other riders, e.g., GMWB [68]. Therefore in recent surveys, there exist no meaningful statistics on these contracts. On the other hand, the GMWB rider is studied in detail in recent surveys [65,69]. Therefore, we choose the attributes of the contracts in the input portfolio, irrespective of the type of the VA contracts, according to the statistics reported for GMWB riders in these survyes.

The study of gender (male/female) distribution for GMWB contracts in [69] shows that males constitute at most 58% (in 1982) and at least 42% (in 2009) of the policyholders. In the most recent survey in 2013, 51% of the policyholders were male and 49% were female. Furthermore, the study doesn't provide any statistics that show dependence between gender and other attributes of the contracts. Therefore, we choose to assign 51% of the contracts to males and 49% to females, independent of the choice of other attributes.

Figure 2-17 in [65] provides a graph of age distribution for GMWB contracts that are in force by the end of 2011. The age of policyholders in the graph ranges from 50

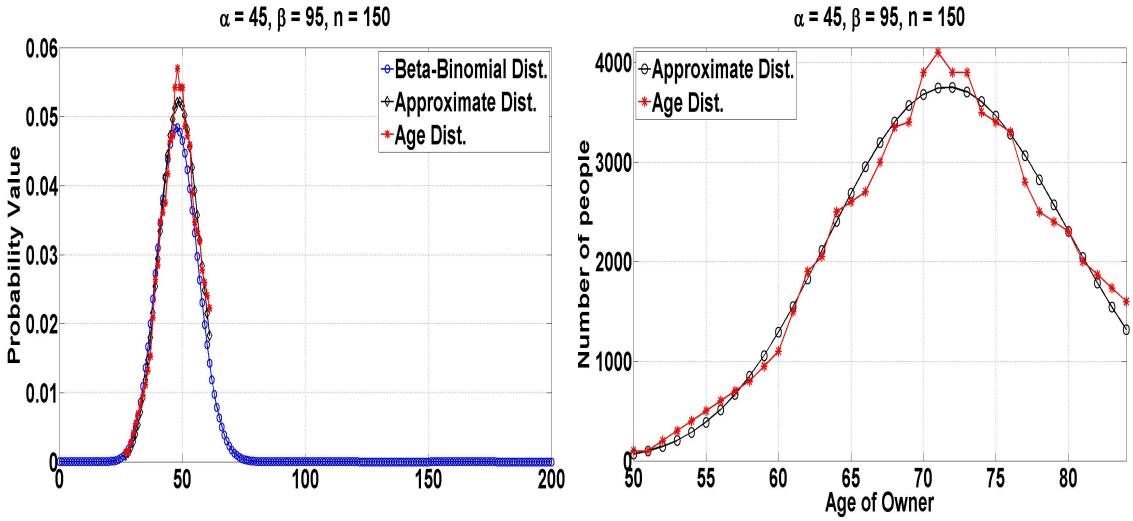


Figure 5.1: Approximate age distribution of Figure 2-17 in [65] and its approximation by part of a rescaled beta-binomial distribution.

years to 84 years. The graph is left skewed and has a fat right tail. The information of Figure 2-17 in [65] is consistent with the age statistics reported in [69] and provides more detailed statistics of age. Thus, we choose to work with the data of Figure 2-17 in [65]. Using a suitable choice of β and α parameters, a beta-binomial distribution can satisfy these properties. We choose to approximate the graph of Figure 2-17 in [65] by part of a beta-binomial distribution with $\alpha = 45, \beta = 95$ and $n = 150$ that is re-scaled to be a probability distribution (black curve in Figure 5.1). The data of [65] indicates that the age of the policyholder can partially explain the amount of the account value. Furthermore, it suggests that there are statistical dependencies between the guarantee value and the account value. However, it provides no significant information that suggests the existence of a statistical dependency between the age of the policyholder and other attributes of the account. In order to best model these dependencies, we assume that policyholders of the VA accounts in the input portfolio have an age distribution that is drawn from the aforementioned approximate distribution of age by part of the rescaled beta-binomial distribution, independently of other attributes.

Given the aforementioned age distribution, we choose the account values between

$5,000 to $330,000 according to the following age specific distribution. For people who are less than 54 years old, we choose the account values uniformly at random in [5,000, 100,000] with a probability of 0.59 and we choose the account values uniformly at random in (100,000, 330,000] with a probability of 0.41. For people whose age is in the range [54, 71], we choose the account values uniformly at random in [5,000, 100,000] with a probability of 0.35 and we choose the account values uniformly at random in (100,000, 330,000] with a probability of 0.65. For people who are older than 71 years old, we choose the account values uniformly at random in [5,000, 100,000] with a probability of 0.47 and we choose the account values uniformly at random in (100,000, 330,000] with a probability of 0.53. These distributions of account values allow us to exactly replicate the age specific statistics that are documented in [65] and fairly accurately estimate the age agnostic statistics that are documented on account values.

The data in [65] indicates that 57% of GMWB contracts have guaranteed values that exceed the account value. Furthermore, the average of account values is 95% of the average of guarantee values. In order to create a skewed distribution that conforms to these statistics, we choose to model the guarantee values for each account as follows.

$$GV = AV + \epsilon$$

where ϵ has a normal distribution with mean $\mu = \frac{AV \times 5}{95}$ and standard deviation $\sigma = \frac{-\mu}{\Phi^{-1}(0.43)} \cong 5.6698\mu$. [4] These choices of parameters are required so that, for each contract, the guarantee value is greater than the account value with a probability of 57% and $\frac{AV}{\mathbb{E}[GV]} = 0.95$. There exists a possibility that for some contracts with small AV values, the realized value of GV is less than zero. For these contracts, we set the value of GV to $10 so that we can keep the contract in force. This modification affected only 19 of the contracts in the input portfolio of these experiments, which is insignificant compared to the size of the input portfolio. This model creates more dependency than the data of [65]

[4] $\Phi(\cdot)$ is the CDF of normal distribution.

indicates, but we choose to work with it so that we can stress test, to some extent, the proposed sampling method. As in our previous experiments, for GMWB contracts, we set the value of the guaranteed death benefit to be equal to the value of the guaranteed withdrawal benefit.

The survey of [65] provides more insights into the withdrawal rates offered by the insurance companies. According to the data in [65], participant insurance companies, which includes major insurance companies, do not offer a wide selection of withdrawal rates to their costumers. The majority of contracts issued by these companies have a withdrawal rate of 7%. These companies also offer withdrawal rates of 5% and 10%. Some of these companies also offer a withdrawal rate of 6%; however, the sale of these contracts was statistically insignificant ($\leq 1\%$) and hence we choose to ignore them. The sale of contracts with withdrawal rate of 5% was common in 2005-2008. However, in recent years (after 2010) contracts with 10% withdrawal rate are common. The data of survey [65] does not show the existence of any statistical dependence between the withdrawal rate and other attributes. Moreover even if there exists a statistical dependence, it should be a weak dependence, as the majority of contracts have a withdrawal rate of 7%. Therefore to incorporate all these statistics in our input portfolio, we choose to assign withdrawal rates to contracts with a GMWB+GMDB rider independently at random and according to the following distribution: withdrawal rate of 5%, 7%, and 10% are assigned with respective probabilities of 0.16714, 0.73714, and 0.09572. These probabilities represent the average share of sales over 2005-2011.

We were not able to find any statistics on the maturity time of annuity contracts. Therefore, to have a fair comparison between the results of this section and the previous section, we choose to assign maturity times independent of other attributes and uniformly at random as integers in the range $[10, 25]$.

Tables 5.9 and 5.10 contain the correlation coefficients and RDC values for each pair of attributes in the input portfolio that we use in this set of experiments. Given the above-

Attributes	Corr. Coef.	Attributes	Corr. Coef.	Attributes	Corr. Coef.
(Rider, Gender)	0.0013	(Rider, Age)	-0.0086	(Rider, AV)	0.0011
(Rider, GV)	0.0010	(Rider, WR)	0.9515	(Rider, Mat)	0.0006
(Gender, Age)	-0.0015	(Gender, AV)	-0.0047	(Gender, GV)	-0.0050
(Gender, WR)	0.0017	(Gender, Mat)	0.0010	(Age, AV)	-0.0800
(Age, GV)	-0.0698	(Age, WR)	-0.0098	(Age, Mat)	-0.0013
(AV, GV)	0.8875	(AV, WR)	0.0015	(AV, Mat)	-0.0026
(GV, WR)	0.0012	(GV, Mat)	-0.0020	(WR, Mat)	0.0011

Table 5.9: Correlation coefficient between each pair of attributes in the synthetic non-uniform input portfolio.

mentioned design of the input portfolio, we expect age, AV and GV to show significant statistical dependency. However, except for rider and withdrawal rate, we don't expect any tangible statistical dependency between any other pair of attributes. The data in Tables 5.9 and 5.10 confirm our hypothesis. The only significant values in these tables are the ones corresponding to pairs of attributes from age, AV and GV as well as the pair of attributes rider and withdrawal rate. Unlike previous input portfolios, there is a more meaningful dependency between rider and withdrawal rate attributes in the input portfolio of this section. Because GMDB contracts are assigned a withdrawal rate of 0 and the majority of contracts with GMWB + GMDB riders are assigned a withdrawal rate of 7%.

We choose the parameters of the neural network to be similar to the parameters of Section 3.2, except that we choose a learning rate of 0.1 and we fit the smoothed MSE values with a polynomial of degree 2. In all of the experiments, we use the end points in Table 5.11 to generate the sample portfolio and we use the end points of Table 5.12 to generate the training portfolio. These end points are almost equally distanced and hence should naturally work with the uniform sampling method. We use 1000 iterations

Attributes	RDC	Attributes	RDC	Attributes	RDC
(Rider, Gender)	0.0013	(Rider, Age)	0.0109	(Rider, AV)	0.0055
(Rider, GV)	0.0097	(Rider, WR)	1.0000	(Rider, Mat)	0.0074
(Gender, Age)	0.0066	(Gender, AV)	0.0098	(Gender, GV)	0.0074
(Gender, WR)	0.0055	(Gender, Mat)	0.0045	(Age, AV)	0.1129
(Age, GV)	0.1033	(Age, WR)	0.0125	(Age, Mat)	0.0084
(AV, GV)	0.9493	(AV, WR)	0.0083	(AV, Mat)	0.0074
(GV, WR)	0.0097	(GV, Mat)	0.0069	(WR, Mat)	0.0093

Table 5.10: Randomized dependence coefficient (RDC) with $k = 10$ and $s = 1/10$ between each pair of attributes in the synthetic non-uniform input portfolio.

Attribute	Value
Guarantee Type	{GMDB, GMDB + GMWB}
Gender	{Male, Female}
Age	$\{50, 54, 58, 62, 67, 71, 75, 79, 84\}$
Account Value	$\{5e3, 5e4, 1e5, 1.5e5, 2e5, 2.5e5, 3.3e5\}$
Guarantee Value	$\{1e3, 0.5e5, 1e5, 1.5e5, 2e5, 2.5e5, 3e5, 3.5e5, 4e5, 4.5e5, 5e5\}$
Withdrawal Rate	$\{0.05, 0.1\}$
Maturity	$\{10, 15, 20, 25\}$

Table 5.11: Attribute values from which representative contracts are generated for experiments.

in the proposed sampling method to come up with the representative contracts and the training portfolio.

Table 5.13 contains the performance statistics of the proposed neural network of Chapter 3 for different choices of sampling method. We do 6 independent runs, each with a different choice of the sample/training/validation portfolio, for each of the sampling

Attribute	**Values**
Guarantee Type	{GMDB, GMDB + GMWB}
Gender	{Male, Female}
Age	$\{52, 56, 60, 64, 69, 73, 77, 81\}$
Account Value	$\{2.5e4, 7.5e4, 1.25e5, 1.75e5, 2.25e5, 2.9e5\}$
Guarantee Value	$\{2.5e4, 7.5e4, 1.25e5, 1.75e5, 2.25e5, 2.75e5, 3.25e5, 3.75e5, 4.25e5, 4.75e5\}$
Withdrawal Rate	$\{0.06, 0.07, 0.08\}$
Maturity	$\{12, 13, 17, 18, 22, 23\}$

Table 5.12: Attribute values from which training contracts are generated for experiments.

Sampling Method	Relative Error (%)		Running Time	
	Mean	STD	Mean	STD
Uniform Sampling	-2.84	8.92	364	146
Proposed Sampling Without Sobol Rand	-2.03	5.03	560	271
Proposed Sampling With Sobol Rand	-1.87	2.78	518	155

Table 5.13: Statistics on the running time and accuracy of the neural network framework when used with the uniform sampling method and the sampling method proposed in Section 5.1 with and without the Sobol quasi-random number generator to estimate the delta value of a non-uniformly distributed input portfolio. The recorded errors are relative errors as defined in (3.19). All times are in seconds.

methods. The statistics in Table 5.13 report the outcome of these runs.

The proposed non-uniform sampling method, incorporating Sobol quasi-random numbers, is the most accurate of the three methods tested. The results also show that both versions of the proposed non-uniform sampling method are more accurate than the uniform sampling method. Also note that the errors reported in Table 5.13 are larger than those in Tables 5.7 and 5.8. Moreover, the increase in the percentage of the relative

error is more significant for the uniform sampling method than for the two non-uniform sampling methods. In particular, the value of the standard deviation for the uniform sampling method is so big that it makes this method useless in practical applications. Although the relative errors are large for both versions of the non-uniform sampling method, these methods are more likely to provide absolute relative error values that are less than the industry accepted error of 5%. In particular, 5 out of the 6 runs for the proposed sampling method with Sobol quasi-random numbers have an absolute relative error value of less than 4% and the absolute relative error value of the other experiment was less than 6%. Therefore, we believe that the proposed non-uniform sampling method with Sobol quasi-random numbers can work reliably even with input portfolios that have statistical dependency between their attributes.

The attributes of non-uniform input portfolios fill some parts of the space in which they are defined more densely than other parts. Having statistical dependence in a non-uniform portfolio further reduces the volume of space occupied by the attributes of a non-uniform portfolio. Uniform sampling methods are designed to fill the whole space and hence they are highly likely to produce samples in parts of the space that are not filled (or only sparsely filled) with contracts in the input portfolio. The proposed sampling method reduces the number of samples in the parts of the sample space that are not occupied (or only sparsely occupied) by the input portfolio. However, the proposed sampling method still creates redundant samples because of the independence assumption that it makes. Given that the size of the sample portfolio and the training portfolio in this set of experiments and in the experiments of Sections 5.2.2 and 5.2.3 are equal, creation of these redundant samples may contribute to the decrease in the accuracy of the sampling method.

The results on the running time are mixed. The average running time of the proposed neural network for all choices of the sampling method has decreased. However, the standard deviation for the proposed non-uniform sampling methods is increased compared to

the results in Tables 5.7 and 5.8. The increase in the standard deviation of both versions of the proposed non-uniform sampling method can be explained by the fact that some of sample portfolios generated by this sampling method do not effectively fill the space making it harder for the proposed neural network to find the best choice of parameters. We suggest further investigation of improved variants of the proposed non-uniform sampling method that do consider statistical dependency to address this issue.

Chapter 6

Conclusions and Future Work

In this thesis, we proposed an efficient and accurate neural network framework to find key risk metrics for large portfolios of VA products. As we discuss in Chapter 1, valuing a large portfolio of VA contracts via nested MC simulations, the industry standard methodology, to find key risk metrics is a time consuming process [29, 61]. Recently, Gan and Lin [30, 32] proposed a Kriging framework that ameliorates the computational demands of the valuation process. Both our proposed method and Kriging are examples of a more general framework called spatial interpolation, which works as follows. A small set of representative VA contracts is selected and valued via MC simulations. The values of the representative contracts are then used in a spatial interpolation method that approximates the value of the contracts in the input portfolio as a linear combination of the values of the representative contracts.

We study three of the most prominent spatial interpolation techniques (i.e., Kriging, IDW, RBF) in Chapter 2. Our study of these traditional spatial interpolation techniques highlights the strong dependence of the accuracy of the framework on the choice of distance function used in the estimations. Moreover, none of the traditional spatial interpolation techniques can provide us with all of accuracy, efficiency, and granularity.

In Chapter 3, we propose a neural network implementation of the spatial interpolation

technique that learns an effective choice of the distance function and provides accuracy, efficiency, and granularity. We study the performance of the proposed approach on a synthetic portfolio of VA contracts with GMDB and GMWB riders. Our results in Section 3.2 illustrate the superior accuracy of our proposed neural network approach in estimation of the delta value for the input portfolio compared to the traditional spatial interpolation techniques. Although our numerical experiments are centred around estimation of the delta value, the application of the proposed neural network framework is not limited to the estimation of the delta value. The network can be easily adapted to estimate any of the Greeks. However, for higher-order Greeks, e.g., Gamma, we may need to increase the number of samples and/or to use polynomials of the features with degrees greater than one. Higher-order Greeks usually have a smaller range of values than first order Greeks. Therefore to have sufficient granularity, we may need more representative contracts to capture the changes in values from one location in the sample space to another. Increasing the number of representative contracts typically reduces the distance between these contracts. Therefore, we need the value of the G function to change significantly as we move between two close representative contracts. Such change in the value of the G function can be achieved by using higher order polynomials of the features.

Training of the neural network requires us to introduce two additional sets of sample VA contracts, i.e., the training portfolio and the validation portfolio, compared to the traditional spatial interpolation frameworks. Our experiments in Chapters 3 and 5 show that, for an appropriate choice of sampling method, if each of the aforementioned sample sets is sufficiently large, the particular realization of the sample set does not significantly affect the accuracy and efficiency of the method. However, if too small, the size of each of these sample sets can significantly affect the performance of our proposed neural network approach.

In Chapter 4, we use the proposed neural network approach to estimate, in an efficient and accurate way, the probability distribution of the portfolio loss, which is key

to calculate the Solvency Capital Requirement (SCR). SCR is an integral part of the new regulatory framework of Solvency II. Because of the imprecise language used to describe the standards in Solvency II, many insurance companies struggle to understand and implement the framework.

In recent years, mathematical frameworks for calculation of the SCR have been proposed to address the former issue [4, 21]. Furthermore, Bauer et al. [4] has suggested a nested MC simulation approach to calculate the SCR to address the later issue. The suggested MC approach is computationally expensive, even for one simple insurance contract. Hence, it cannot be easily generalized to a large portfolio of insurance products.

The results of our numerical experiments in Section 4.3 corroborate the superior accuracy and efficiency of the sequential implementation of our proposed neural network approach compared with an implementation of the MC approach that uses parallel processing.

Although our method requires us to train our neural network using three small ($< 1\%$ of the size of the input portfolio) portfolios, as we mention earlier, if these small portfolios are sufficiently large, the performance of the method has low sensitivity to the particular realization of these portfolios.

As we discuss in Chapter 2, one key parameter that can noticeably affect the performance of the spatial interpolation framework is the choice of sampling method. In practice, insurance companies often deal with VA portfolios that are non-uniformly distributed in the space in which the input portfolio is defined. Intuitively, an estimation method can benefit from better accuracy in dense regions of the space. Hence, our sampling method, intuitively, should consider the distribution of the VA policies in the input portfolio. However, as we elaborate in Chapter 3, a bad sample with dense clusters of representative contracts can decrease the accuracy of the proposed neural network by over emphasizing the influence that the value of sample VAs in each of the clusters has on the value of VA policies in the space. Hence, a good sampling method should avoid

creating dense clusters in the sample VA portfolio.

In Chapter 5, we build on the above-mentioned insights and propose a non-uniform sampling method that uses an iterative approach to generate samples that can effectively fill the space and have similar per attribute probability distributions as the input portfolio. To effectively fill the space and avoid duplicate VAs in the generated samples, the proposed sampling method uses a Sobol quasi-random number generator. Our experimental results in Section 5.2 show that the proposed sampling method, if used to create the representative portfolio and the training portfolio, achieves better accuracy results, i.e., lower mean and standard deviation of the recorded relative errors, than the simple uniform sampling method used in previous chapters.

In this thesis, we assume the existence of a single asset that backs all of our VA contracts. Although this assumption allows us to simplify our analysis and implementations, in reality, insurance companies invest, on behalf of policyholders, in a basket of assets. Hence, the evolution of the account value, for each contract, can be better described using a multi-asset model. Multi-asset models have significantly more complex structure for their key risk metrics than single-asset models. For example, in a two asset model, a study of the delta value should consider a surface of delta values rather than a single delta value for the input portfolio.

Increasing the complexity of key risk metrics requires us to revisit the training methodology of the proposed neural network (similar to our study in Chapter 4). Moreover, it might require us to make changes to the structure of neurons in the proposed neural network. Extending our single asset model to a multi-asset model is a possible future research direction.

In our introductory study in Chapter 4, our simple model of financial markets allows us to simply sort the $C_{s_1}^{(i)}$ values that represent market conditions to achieve the best outcome with our neural network approach. However, this simple strategy is not as straightforward for more complex models of the financial markets. Therefore, we need

to study an effective strategy to exploit the closeness of the sample points representing various states of more complex models of financial markets to reduce the training time of the neural network. We leave this study for future work.

The neural network that we used in our experiments was implemented sequentially. Given the structure of parameters/variables that define the behavior of the neural network, we can easily develop a parallel implementation of the neural network. A parallel implementation can significantly reduce the training time of the neural network and thereby the running time of the neural network framework. Hence another line of future work is to develop a parallel implementation of the neural network.

Although, in Chapter 4, we do not study in depth the problem of having a large number of outer simulation scenarios, we proposed a possible solution via data interpolation to alleviate this problem. In our experiments, to reduce the simulation times and because of the smoothness of the curve that describes the Δ values, as a first simple choice, we suggest using piecewise-linear interpolation to approximate the Δ values. However, our experimental results suggest that using a better interpolation method may increase the accuracy of our proposed neural network approach for nested simulations. We suggest a study on the choice of the interpolation scheme as another possible future work.

Our proposed non-uniform sampling method in Chapter 5 assumes statistical independence between the attributes that define the input portfolio. Our numerical results in Section 5.2 show that the performance of the proposed neural network may deteriorate when this assumption does not hold, i.e., the attributes of the input portfolio have strong statistical dependence. Therefore, to further improve the performance of the proposed neural network and decrease its sensitivity to statistical dependencies in the input portfolio, we suggest a further study on ways to incorporate the information on statistical dependencies of the input portfolio into the proposed sampling method.

Appendix A

How To Choose The Training Parameters

The training method that we discuss in Section 3.1 is dependent on the choice of several free parameters such as the learning rate and $\mu_{\max}$. In this appendix, we discuss heuristic ways to choose a value for each of these free parameters and justify each choice.

In order to determine a good choice of the learning rate and the batch size, we need to train the network for some number of iterations, say 3000, and study the error associated with the training portfolio as a function of the number of iterations. If the graph has a general decreasing trend and it does not have many big jumps between consecutive iterations, then we say that the choice of the learning rate/batch size is stable. Otherwise, we call the choice of the learning rate/batch size unstable.

From (3.13), we see that the choice of the learning rate parameter affects the amount of change in the weight and bias parameters per iteration. As we discuss in Section 3.1.2, too small of a change increases the training time while too big of a change causes numerical instability. From Figure A.1, we see that, as we increase the value of the learning rate, the graph of the error values moves downwards which means that the training has sped up. However, for a learning rate equal to 2, we see many big jumps in

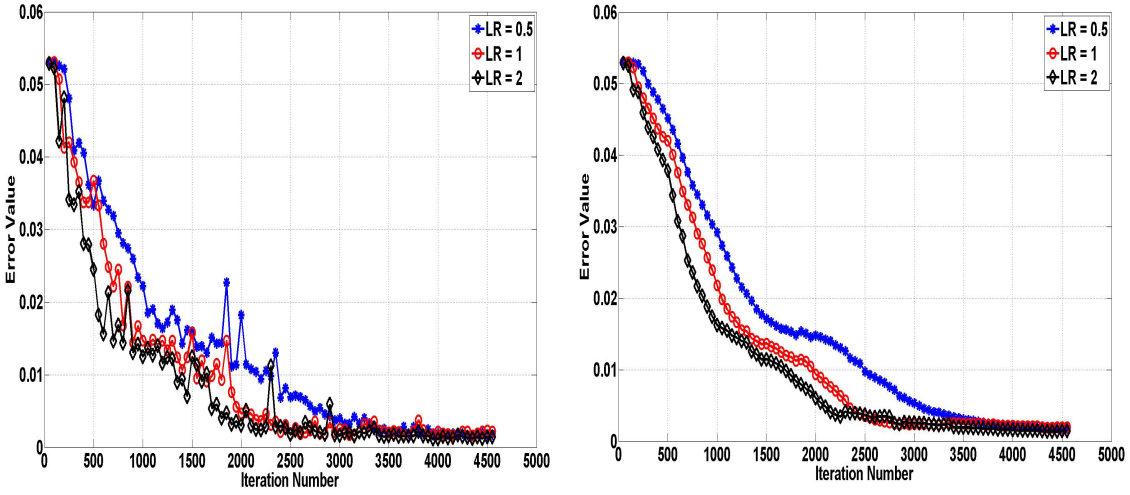


Figure A.1: The MSE error graph (left) and the moving average smoothed MSE error graph (right) of the training portfolio as a function of iteration number and learning rate.

the graph which suggests numerical instability. The numerical instability is more obvious from the moving average smoothed curve of error values. More specifically, starting from iteration 2000, the smoothed MSE error graph for a learning rate of 2 has big jumps which are signs of numerical instability. Note that the smoothed MSE error graphs for learning rates 0.5 and 1 are much smoother.

To find a good choice of the learning rate, we can start from a value of 1 for the learning rate and determine if that choice is stable? If the choice of learning rate is stable, we double the value of the learning rate and repeat the process until we find a learning rate which is unstable. At this point, we stop and choose the last stable value of the learning rate as our final choice of the learning rate. If the learning rate equal to 1 is unstable, we decrease the value of learning rate to half of its current value and repeat this process until we find a stable learning rate.

The batch size controls the speed of training and the amount of error that we get in approximating the gradient of the MSE for the entire training set. Small batch sizes increase the speed of training; however, they also increase the amount of error in approximating the gradient of the MSE error. A good batch size should be small enough to

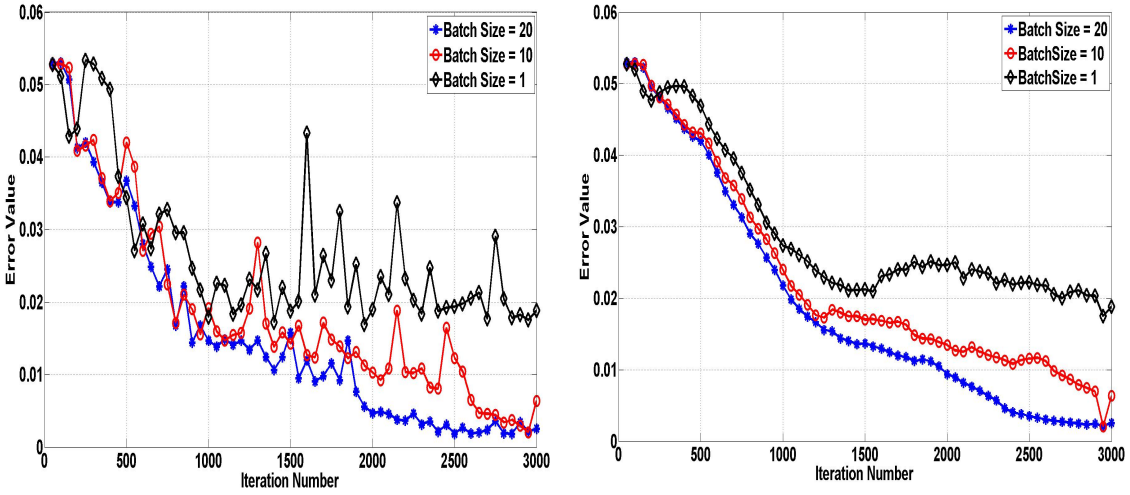


Figure A.2: The MSE error graph (left) and the moving average smoothed MSE error graph (right) of the training portfolio as a function of iteration number and batch-size.

increase the speed of training but not so small as to introduce a big approximation error. To find a good batch size value, we start with a small value, say 5, and determine if this choice of batch size is stable. If so, we stop and choose it as our final choice of the batch size. If the batch size is unstable, we double the batch size and repeat the process until we find a stable batch size.

Figure A.2 shows that small batch size values are associated with many big jumps and hence are unstable. As we increase the batch size value, the error graph becomes much more stable– fewer jumps and a more consistent decreasing trend.

Notice that, in the aforementioned processes for finding the appropriate value of the learning rate and batch size, doubling the values may seem too aggressive as the values may increase or decrease too quickly. To alleviate this problem, upon reaching a desired value, we can do a binary search between the final choice of the parameter's value and the next best choice (the value of parameter before the final doubling) of the parameter's value.

Nesterov, [55, 56], advocates a constant momentum coefficient for strongly convex functions and advocates Equation (A.1) when the function is not strongly convex [67].

$$\mu_t = 1 - \frac{3}{t+5} \tag{A.1}$$

Equation (3.15), suggested in [67], blends a proposal similar to Equations (A.1) and a constant momentum coefficient. Equation (A.1) converges quickly to values very close to 1. In particular, for $t \geq 25$, $\mu_t \geq 0.9$. Hence, as suggested in [67], we should choose a large value (e.g., $0.9, 0.99, 0.995, 0.999$) of $\mu_{\max}$ to achieve better performance and that is what we suggest too.

In Section 3.1.3, we proposed a mechanism to detect stopping events and avoid over-training of the network. As part of this mechanism, we need to record the MSE of the validation set every I^{th} iteration. Too small values of I can slow down the training process while too big values of I can result in losing information regarding the trend that exists in the MSE graph. In order to find a good value of I that neither slows down the training too much nor creates excessive information loss, we can use a multiplicative increase process similar to that described above for the batch size. We start with a small value of I, say 10, and train the network for some 4000 iterations and draw the graph of MSE values. We then double the I value and graph the MSE for the new value of I. If the MSE graph for the new value of I has a similar trend as the MSE graph for the previous value of I, we keep increasing the value of I and repeat the process. But, if the resulting graph has lost significant information regarding increasing/decreasing trends in the previous graph, then we stop and choose the previous value of I as the appropriate choice of I. For example, in Figure A.3, the MSE graph corresponding to the value of 100 has fewer big valleys and big peaks than the MSE graph for the value of 50. Hence, if we were to use 100, rather than 50, for the value of I, we would lose a significant amount of information regarding trends in the graph. However, the MSE graph for the value of I equal to 10 has a roughly similar number of big valleys and big peaks compared with the MSE graph for the value of I equal to 50. Hence, the value of 50 is a much better choice for I than either 100 or 10. The value of 50 allows for a faster training than the

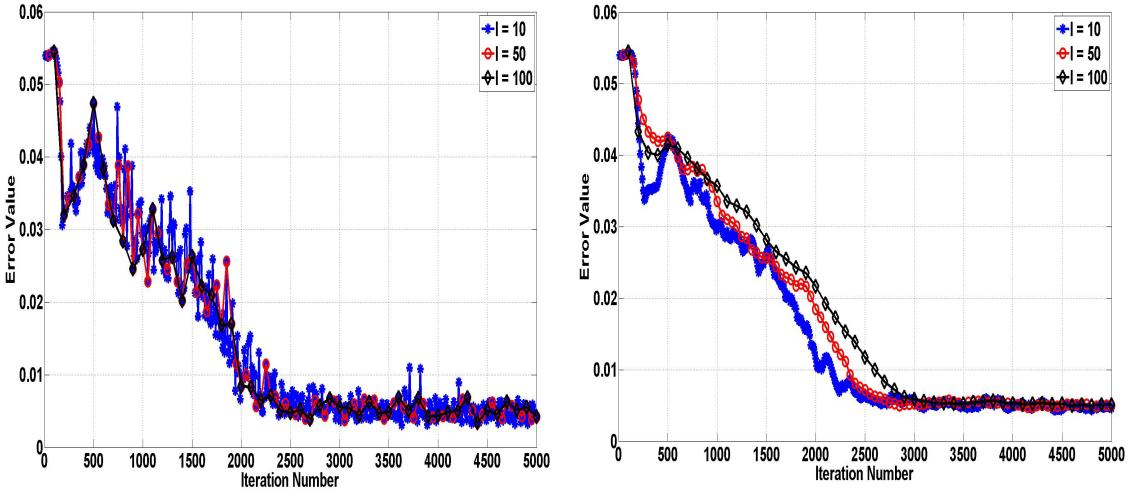


Figure A.3: The MSE error graph (left) and the moving average smoothed MSE error graph (right) of the validation portfolio as a function of iteration number and I value.

value of 10 and retains more information regarding increasing/decreasing trends in the MSE error graph than the value of 100.

We use data smoothing and polynomial fitting to extract the major u-shape trend in the MSE graph and hence find stopping events. In order to find a good choice for the smoothing window, we start with a small value of the smoothing window and calculate the smoothed curve. If the small peaks and valleys of the original curve are suppressed and big peaks and big valleys of the original curve are significantly damped, then we choose that value of the smoothing window as our final choice for the smoothing window. For example, in Figure A.4, the smoothed curve with a smoothing window of 5 has a big valley around iteration number of 400. However the valley is dampened in the smoothed graph resulting from smoothing window of 10.

The primary goal of the polynomial fitting is to find the u-shaped trend in the graph so that we can detect the stopping event. To model the u-shaped trend, we require that the polynomial should go to infinity as its argument goes to either plus or minus infinity. Therefore, the degree of the polynomial should be even. Since we are only interested in detecting a u-shaped trend, it is sufficient to use polynomials of low degree

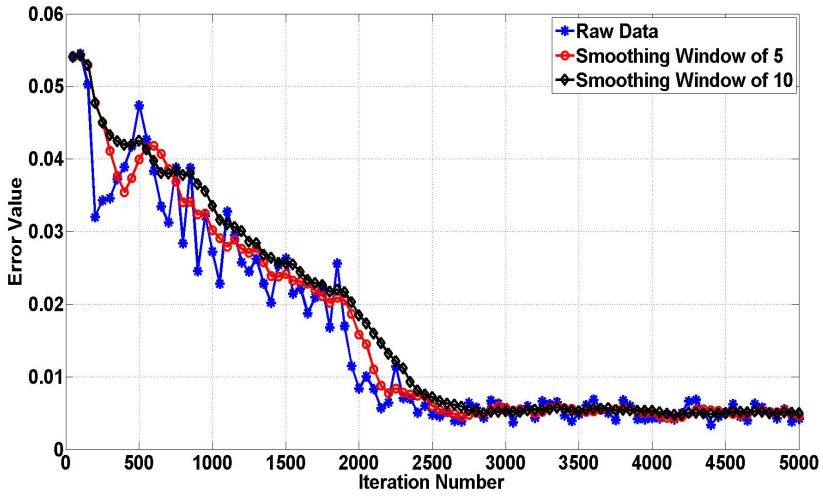


Figure A.4: The MSE error graph of the validation portfolio as a function of iteration number and smoothing window value.

(≤ 10). High degree polynomials overfit the data and they can't detect a slowly increasing trend such as the one in Figure A.4 after iteration 2500. On the other hand, a simple polynomial of degree 2 does not always work well. A quadratic polynomial on a MSE graph similar to Figure A.4 falsely detects a u-shape trend in the big valley between iteration numbers 0 and 500. However a polynomial of degree 4 or higher will not trigger such a false stopping event. Because we smooth the data before we fit any polynomials and we choose our learning parameter such that we expect an initial decreasing trend, we suggest polynomials of degree $4, 6$ or 8 to be used to fit the data to find u-shaped trends.

Finally for the value of window length to detect that we have reached the minimum, we choose a value of W such that the number of iterations in the window ($W \times I$) is big enough (around a hundred iterations) that we can confidently say the graph of the MSE error has reached a minimum value and started to increase in value (an increasing trend). Notice that the window length should not be too big so that we can start the search in the local neighborhood and minimize the training time.

Bibliography

[1] https://docs.oracle.com/javase/7/docs/api/java/util/Random.html.

[2] P.A. Azimzadeh and P.A. Forsyth. The Existence of Optimal Bang-Bang Controls for GMxB Contracts. *SIAM Journal on Financial Mathematics*, 6(1):117–139, 2015.

[3] D. Bauer, A. Kling, and J. Russ. A Universal Pricing Framework For Guaranteed Minimum Benefits in Variable Annuities. *ASTIN Bulletin*, 38:621–651, 2008.

[4] Daniel Bauer, Andreas Reuss, and Daniela Singer. On the Calculation of the Solvency Capital Requirement Based on Nested Simulations. *ASTIN Bulletin*, 42:453–499, November 2012.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, NJ, USA, 2006.

[6] Shyam Boriah, Varun Chandola, and Vipin Kumar. Similarity Measures for Categorical Data: A Comparative Evaluation. In *In Proceedings of the eighth SIAM International Conference on Data Mining*, pages 243–254, 2008.

[7] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, NY, USA, 2004.

[8] P. Boyle and W. Tian. The Design of Equity-Indexed Annuities. *Insurance: Mathematics and Economics*, 43:303–315, 2008.

[9] Phelim P. Boyle and Mary R. Hardy. Reserving for Maturity Guarantees: Two Approaches. *Insurance: Mathematics and Economics*, 21:113–127, 1997.

[10] Paul Bratley and Bennett L. Fox. Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator. *ACM Trans. Math. Softw.*, 14(1):88–100, March 1988.

[11] Mark Broadie, Yiping Du, and Ciamac C. Moallemi. Risk Estimation Via Regression. *Operations Research*, 63:1077–1097, 2015.

[12] P.A. Burrough, R.A. McDonnell, and C.D. Lloyd. *Principles of Geographical Information Systems*. Oxford University Press, 2nd edition, 1998.

[13] A. Belanger, P. Forsyth, and G. Labahn. Valuing the Guaranteed Minimum Death Benefit Clause with Partial Withdrawals. *Applied Mathematical Finance*, 16(6):451–496, 2009.

[14] J. Carriere. Valuation of the Early-Exercise Price for Options Using Simulations and Nonparametric Regression. *Insurance: Mathematics and Economics*, 19(1):19–30, 1996.

[15] M. Cathcart and S. Morrison. Variable Annuity Economic Capital: the Least-Squares Monte Carlo Approach. *Life & Pensions*, pages 36–40, October 2009.

[16] CEIOP. EIOPA Report on the Fifth Quantitative Impact Study (QIS5) for Solvency II. 2011.

[17] Z. Chen and P.A. Forsyth. A Numerical Scheme for the Impulse Control Formulation of Pricing Variable Annuities with a Guaranteed Minimum Withdrawal Benefit (GMWB). *Numerische Mathematik*, 109:535–569, June 2008.

[18] Z. Chen, K. Vetzal, and P.A. Forsyth. The Effect of Modelling Parameters on the Value of GMWB Guarantees. *Insurance: Mathematics and Economics*, 43:165–173, 2008.

[19] Y. Chi and X. S. Lin. Are Flexible Premium Variable Annuities Underpriced? *ASTIN Bulletin*, 42(2):559–574., 2012.

[20] J. P. Chiles and P. Delfiner. *Geostatistics, Modelling Spatial Uncertainty*. Wiley-Interscience, 1999.

[21] Marcus C. Christiansen and Andreas Niemeyer. Fundamental Definition of the Solvency Capital Requirement in Solvency II. *ASTIN Bulletin*, 44(3):501–533, 2014.

[22] T. Coleman, Y. Li, and Maria-Cristian Patron. Hedging Guarantees in Variable Annuities under Both Equity and Interest Rate Risks. *Insurance: Mathematics and Economics*, 38:215–228, 2006.

[23] Noel A. C. Cressie. *Statistics for Spatial Data*. John Wiley & Sons, Inc., New York, USA, 1993.

[24] M. Dai, Y. K. Kwok, and J. Zong. Guaranteed Minimum Withdrawal Benefit in Variable Annuities. *Journal of Mathematical Finance*, 18(4):595–611, 2008.

[25] S. Daul and E. Vidal. Replication of Insurance Liabilities. *RiskMetrics Journal*, 9(1):79–96, 2009.

[26] R. Dembo and D. Rosen. The Practice of Portfolio Replication: A Practical Overview of Forward and Inverse Problems. *Annals of Operations Research*, 85:267–284, 1999.

[27] Y. d'Halluin, P. Forsyth, and K. Vetzal. Robust Numerical Methods for Contingent Claims Under Jump Diffusion Processes. *IMA Journal of Numerical Analysis*, 25:65–92, 2005.

[28] V. A. Epanechnikov. Non-Parametric Estimation of a Multivariate Probability Density. *Theory of Probability and Its Applications*, 14(1):153–158, 1967.

[29] J. Fox. A Nested Approach to Simulation VaR Using MoSes. *Insights: Financial Modelling*, pages 1–7, 2013.

[30] G. Gan. Application of Data Clustering and Machine Learning in Variable Annuity Valuation. *Insurance: Mathematics and Economics*, 53(3):795–801, 2013.

[31] G. Gan. Representative Variable Annuity Policy Selection using Latin Hypercube Sampling. November 2013.

[32] G. Gan and X. S. Lin. Valuation of Large Variable Annuity Portfolios Under Nested Simulation: A Functional Data Approach. *Insurance: Mathematics and Economics*, 62(0):138–150, 2015.

[33] G. Gan, C. Ma, and J. Wu. *Data Clustering: Theory, Algorithms and Applications*. SIAM Press, Philadelphia, PA, USA, 2007.

[34] H. Gerber, E. Shiu, and H. Yang. Valuing Equity-Linked Death Benefits and Other Contingent Options: A Discounted Density Approach. *Insurance: Mathematics and Economics*, 51(1):73–92, 2012.

[35] L. Girard. An Approach to Fair Valuation of Insurance Liabilities Using the Firm's Cost of Capital. *North American Actuarial Journal*, 6:18–41, 2002.

[36] M. Hardy. *Investment Guarantees: Modeling and Risk Management for Equity-Linked Life Insurance*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2003.

[37] S. A. Hejazi and K. R. Jackson. A Neural Network Approch to Efficient Valuation of Large Portfolios of Variable Annuities. *Insurance: Mathematics and Economics*, 70:169–181, 2016.

[38] S. A. Hejazi and K. R. Jackson. Effcient Valuation of SCR via a Neural Network Approach. Manuscript Submitted for Publication to

the Journal of Insurance: Mathematics and Economics, Available at http://www.cs.toronto.edu/pub/reports/na/IME-Paper3.pdf, 2016.

[39] S. A. Hejazi, K. R. Jackson, and G. Gan. A Spatial Interpolation Framework for Efficient Valuation of Large Portfolios of Variable Annuities. Manuscript Submitted for Publication to the Journal of Insurance: Mathematics and Economics, Available at http://www.cs.toronto.edu/pub/reports/na/IME-Paper1.pdf, 2015.

[40] K. Hornik. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks*, 4(2):251–257, 1991.

[41] Y. Huang and P.A. Forsyth. Analysis of A Penalty Method for Pricing a Guaranteed Minimum Withdrawal Benefit (GMWB). *IMA Journal of Numerical Analysis*, 32(1):320–351, June 2011.

[42] John C. Hull. *Options, Futures, and Other Derivatives*. Pearson Prentice Hall, Upper Saddle River, NJ, 6th edition, 2006.

[43] IRI. The 2011 IRI Fact Book. *Insured Retirement Institute*, 2011.

[44] S. Joe and F. Y. Kuo. Constructing Sobol Sequences with Better Two-Dimensional Projections. *SIAM Journal on Scientific Computing*, (30):2635–2654, 2008.

[45] D.G. Krige. *A Statistical Approach to Some Mine Valuations and Allied Problems at the Witwatersrand*. Master's thesis, University of Witwatersrand, 1951.

[46] X. Lin, K. Tan, and H. Yang. Pricing Annuity Guarantees under a Regime-Switching Model. *North American Actuarial Journal*, 13:316–338, 2008.

[47] F. Longstaff and E. Schwartz. Valuing American Options by Simulation: A Simple Least-Squares Approach. *The Review of Financial Studies*, 14(1):113–147, 2001.

[48] David Lopez-Paz, Philipp Hennig, and Bernhard Schlkopf. The Randomized Dependence Coefficient. Available at http://arxiv.org/abs/1304.7717, 2013.

[49] G. Matheron. Principles of Geostatistics. *Economic Geology*, 58:1246–1266, 1963.

[50] M. McKay, R. Beckman, and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239–245, 1979.

[51] M. Milevsky and T. Salisbury. Financial Valuation of Guaranteed Minimum Withdrawal Benefits. *Insurance: Mathematics and Economics*, 38:21–38, 2006.

[52] T. Moenig and D. Bauer. Revisiting the Risk-Neutral Approach to Optimal Policyholder Behavior: A study of Withdrawal Guarantees in Variable Annuities. In *12th Symposium on Finance, Banking, and Insurance*, Germany, December 2011.

[53] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[54] E. A. Nadaraya. On Estimating Regression. *Theory of Probability and its Applications*, 9:141–142, 1964.

[55] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Applied Optimization. Springer US, 2003.

[56] Yurii Nesterov. A Method of Solving a Convex Programming Problem with Convergence Rate O(1/sqrt(k)). *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[57] J. Oechslin, O. Aubry, M. Aellig, A. Kappeli, D. Bronnimann, A. Tandonnet, and G. Valois. Replicating Embedded Options in Life Insurance Policies. *Life & Pensions*, pages 47–52, 2007.

[58] E. Parzen. On Estimation of a Probability Density Function and Mode. *Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[59] B. T. Polya. Some Methods of Speeding up the Convergence of Iteration Methods. *USSR Computational Mathematics and Mathematical Physics*, 4:1–17, 1964.

[60] Alfred Renyi. On Measure of Dependence. *Acta Mathematica Academiae Scientiarum Hungaricae*, 10:441–451, 1959.

[61] C. Reynolds and S. Man. Nested Stochastic Pricing: The Time Has Come. *Product Matters!— Society of Actuaries*, 71:16–20, 2008.

[62] Jurgen Schmidhuber. Deep Learning in Neural Networks: An Overview. 2014.

[63] Donald Shepard. A Two-dimensional Interpolation Function for Irregularly-spaced Data. In *Proceedings of the 1968 23rd ACM National Conference*, pages 517–524, NY, USA, 1968. ACM.

[64] I. M. Sobol'. On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

[65] Society of Actuaries and LIMRA. *Variable Annuity Guaranteed Living Benefits Utilization Study-2013 Experience.* 2013. Available at https://www.soa.org/Files/Research/research-2016-limra-vaglbus-2013-experience.pdf.

[66] E.M. Stein and R. Shakarchi. *Real Analysis: Measure Theory, Integration, and Hilbert Spaces.* Princeton University Press, 2009.

[67] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On The Importance of Initialization and Momentum in Deep Learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1139–1147. JMLR Workshop and Conference Proceedings, May 2013.

[68] TGA. Variable Annuities—An Analysis of Financial Stability. *The Geneva Association*, March 2013.

[69] The Gallup Organization and Mathew Greenwald & Associates. *2013 Survey of Owners of Individual Annuity Contracts*. The Committee of Annuity Insurers, 2013. Available at http://www.annuity-insurers.org/wp-content/uploads/2013/10/2013-Gallup-Survey.pdf.

[70] E.R. Ulm. The Effect of the Real Option to Transfer on the Value Guaranteed Minimum Death Benefit. *The Journal of Risk and Insurance*, 73(1):43–69, 2006.

[71] J. Vadiveloo. Replicated Stratified Sampling—A New Financial Modelling Option. *Tower Watson Emphasis Magazine*, pages 1–4, 2011.

[72] Geoffrey S. Watson. Smooth Regression Analysis. *Sankhyā: Indian Journal of Statistics*, 26:359–372, 1964.