Resilient Machine Learning Approaches for Fast Risk Evaluation and Management in Financial Portfolios and Variable Annuities

by

Xintong Li

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Actuarial Science

Waterloo, Ontario, Canada, 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

Supervisor(s):	Mingbin Feng Associate Professor, Dept. of Statistics and Actuarial Science University of Waterloo
	Tony S. Wirjanto Professor, Dept. of Statistics and Actuarial Science University of Waterloo
Internal Member:	Mary R. Hardy Professor, Dept. of Statistics and Actuarial Science University of Waterloo
	Chengguo Weng Professor, Dept. of Statistics and Actuarial Science University of Waterloo
Internal-External Member:	Yuying Li Professor, School of Computer Science University of Waterloo
Other Member(s):	Emiliano Valdez Professor, Dept. of Mathematics University of Connecticut

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

Part of the work in this thesis has been published in a Winter Simulation Conference proceeding.

I understand that my thesis may be made electronically available to the public.

Abstract

Risk management of financial derivatives and actuarial products is intricate and often requires modeling the underlying stochasticity with Monte Carlo simulations. Monte Carlo simulation is flexible and can easily adapt to changes in model assumptions and market conditions. However, as multiple sources of risk are considered over long time horizons, the simulation model becomes complex and time-consuming to run. Tremendous research effort has been dedicated to designing computationally efficient machine learning-based procedures that mitigate the computational burden of a standard simulation procedure. In machine learning, model flexibility comes at the expense of model resilience, which is crucial for risk management tasks. This study considers estimating tail risks of complex financial and actuarial products with resilient machine learning-based nested simulation procedures. We propose a novel metamodeling approach that integrates deep neural networks within a nested simulation framework for efficient risk estimation. Our approaches offer substantial improvements over the associated standard simulation procedures. This study also illustrates how to build and assess resilient machine learning models for different problem complexities and different data structures, qualities, and quantities. To further enhance adaptability to new variable annuity contracts and changing market conditions, this thesis explores transfer learning techniques. By reusing and fine-tuning pre-trained metamodels, the proposed approach accelerates the adaptation process to different contract features and evolving market dynamics without retraining models from scratch. Transfer learning improves computational efficiency and enhances the robustness and flexibility of neural network metamodels in dynamic hedging of variable annuities.

Extensive numerical experiments in this thesis demonstrate that the proposed methods substantially improve computational efficiency, sometimes shortening runtime by orders of magnitude compared to standard nested simulation procedures. The results indicate that deep neural network metamodels with transfer learning can quickly adapt to new market scenarios and contract specifications. This research contributes to the advancement of risk management practices for complex actuarial products and financial derivatives. By leveraging advanced machine learning techniques, this thesis offers a practical and scalable solution for insurers to perform timely and accurate risk assessments. The integration of long short-term memory metamodels and transfer learning into a nested simulation framework represents a major step forward toward more efficient, adaptable, and robust methodologies in actuarial science and quantitative finance.

Acknowledgements

I would like to express my deepest gratitude to Professor Mingbin Feng and Professor Tony Wirjanto for their invaluable support and guidance throughout my academic journey. Their unwavering patience and profound wisdom have been instrumental in helping me navigate the complexities of my research. I am deeply grateful for their mentorship, which has not only sharpened my analytical skills but also inspired me to pursue excellence in my work.

Additionally, I extend my sincere thanks to my best friends. Their unwavering support and encouragement have been a constant source of strength that has helped me overcome obstacles and maintain my motivation throughout this process. Words cannot fully convey how much their presence has meant to me over these years.

I also wish to acknowledge the support of my family, who have always been a source of love and encouragement. They have provided me with the emotional stability I needed to focus on my studies and research.

Table of Contents

List of Tables

List of Figures

Abbreviations

AE Absolute Error 19, 25–31, 47

ARCH Autoregressive Conditional Heteroskedasticity 110

ARIMA Autoregressive Integrated Moving Average 110

CNN Convolutional Neural Network 69, 88

CVaR Conditional Value at Risk 20, 21, 23, 25, 28, 38, 44, 48, 56, 58, 59, 71, 73, 79, 80, 82–84, 86, 91–95, 100–105

DDPG Deep Deterministic Policy Gradient 144

DNN Deep Neural Network 10, 68–73, 80, 82, 86–88, 90, 92, 95, 99, 105–112, 134, 144

DRL Deep Reinforcement Learning 134, 135, 139

FNN Feedforward Neural Network 8–11, 86–89, 91, 94, 103

GARCH Generalized Autoregressive Conditional Heteroskedasticity 110

GBM Geometric Brownian Motion 76, 110, 120, 144, 151, 153–155

GLM Generalized Linear Model 7

GLWB Guaranteed Lifetime Withdrawal Benefit 74

GMAB Guaranteed Minimum Accumulation Benefit 74, 93

GMDB Guaranteed Minimum Death Benefit 74

GMMB Guaranteed Minimum Maturity Benefit x, xii, 3, 74, 75, 117, 120–130, 132, 135, 138, 140, 145, 151, 152, 155, 156

GMWB Guaranteed Minimum Withdrawal Benefit x, xiii, 3, 74–77, 79, 84, 87, 93, 102, 117, 120, 126–130, 132, 135, 138–140, 155, 156

kNN K-Nearest Neighbors 24, 25, 27, 40–43, 48, 52, 53, 62–66, 71

KRR Kernel Ridge Regression 17, 24, 25, 48, 71

LSTM Long Short-Term Memory 9–12, 14, 15, 68, 72, 86–89, 91–106, 108–133, 147–149, 152, 153

MC Monte Carlo 16, 18, 21, 22, 32, 36, 37, 44, 109

MDP Markov Decision Process 136, 139, 140, 143

ML Machine Learning 108

MLMC Multi-Level Monte Carlo 17, 21, 22, 27, 28, 45, 50, 60, 61

MLR Multiple Linear Regression 86, 87, 91

MSE Mean Squared Error x, 6, 7, 16, 17, 19, 21, 25–33, 38, 40, 44, 47, 49, 50, 53–58, 60, 87, 91, 99, 113, 116, 121, 122, 125

P&L Profit and Loss 77–79

PPO Proximal Policy Optimization x, xiii, 145–149, 151–157

QPR Quadratic Programming Regression 86, 87, 91, 93

ReLU Rectified Linear Unit 9

RKHS Reproducing Kernel Hilbert Space 25

RL Reinforcement Learning 134–136, 140–144, 146, 155, 156

RNN Recurrent Neural Network 11, 68, 86–89, 93, 94, 109, 110

RRMSE Relative Root Mean Squared Error 103–105

RS-GBM Regime-Switching Geometric Brownian Motion xii, xiii, 76, 110, 120–124, 126, 128, 130, 153

SGD Stochastic Gradient Descent 13

TL Transfer Learning x, 109–113, 117, 118, 120–122, 124–129, 131, 133, 135

TRPO Trust Region Policy Optimization 145, 146

VA Variable Annuity x, 1–3, 14, 15, 70–75, 77–79, 84, 105–114, 117–121, 126, 128–133, 135–140, 144, 151, 155–157

VaR Value at Risk 19–21, 23, 25, 28, 31, 38, 44, 47, 48, 50, 51, 53, 56–59, 63–66, 73

Chapter 1

Introduction

Risk management is a key component of modern financial systems. Successful risk management practice can enhance financial stability and resilience for individual companies and for the financial system as well. VA contracts are a popular type of complex insurance product that is linked to the performance of underlying assets. They provide a guaranteed minimum income benefit to the policyholder regardless of the performance of the underlying assets. Advanced Monte Carlo simulation techniques, particularly nested simulation procedures, become indispensable for risk assessment of such products. In contrast to finite-difference methods, a Monte Carlo simulation scheme is more flexible and can be easily adapted to model tail risk with a rule-based design. Glasserman (2004) provide a comprehensive overview of Monte Carlo simulation methods in financial engineering and risk management applications. In this thesis, we focus on building and analyzing nested simulation procedures for risk management applications of financial derivatives and insurance products. Nested simulation, also known as nested stochastic modeling and stochastic-on-stochastic modeling, becomes a necessary tool when stochastic simulation of a parameter of interest is contingent on another quantity to be determined stochastically. In the context of financial engineering, nested simulation is used to model the tail risk of a contract whose payoff depends on a set of underlying risk factors. For example, estimating the value of an exotic option over a risk horizon requires simulation given a realization of the underlying assets over that horizon. A standard nested simulation procedure consists of two levels of simulation: the outer-level simulation generates the underlying risk factors, while the inner-level simulation estimates the value of interest with the inner sample mean from another level of Monte Carlo simulation. This nested structure allows for accurate estimation given sufficient computational resources, but it also introduces additional complexity in the simulation design and implementation. Furthermore, in real-world applications, the

computational burden of a nested simulation can be prohibitive. The number of inner simulations required to achieve a desired level of accuracy for each outer scenario can be large. Metamodeling techniques can reduce the computational burden of nested simulations by approximating the inner simulation model. Metamodels are statistical models that approximate the output of a complex simulation model as a function of its input parameters. In this thesis, we focus on metamodels of the inner simulation models in nested simulation procedures for risk management applications of financial derivatives and VA contracts. This thesis addresses two key problems that necessitate nested simulation:

- estimating the risk of a portfolio of financial options and
- dynamic hedging of VA contracts with a delta hedging strategy.

Risk management of VAs is a challenging problem due to the complex interactions between the policyholder's behavior and the financial market dynamics. We focus mainly on the estimation of tail risk measures of VA contract losses using metamodel-based nested simulation procedures. This thesis consists of four chapters. Chapter 2 summarizes theoretical convergence results of several state-of-the-art single-period nested simulation procedures under the same analytical framework. Numerical experiments are conducted to test their empirical convergence behavior under finite budget sizes. Chapter 3 proposes the use of neural network models as metamodels for a two-stage multi-period nested simulation procedure on VA contracts. In our numerical experiment, the best neural network metamodel surpasses the state-of-the-art metamodels in tail identification, and it leads to substantial computational savings. Chapter 3 also includes sensitivity testing of the neural network metamodel. We argue that for estimating tail risk measures of VA contract losses, the inner simulation can be replaced entirely by a suitable metamodel. Eliminating the inner simulation can lead to more substantial computational savings without affecting estimation quality. When extensive inner simulations are required for regulatory purposes, effective budget allocation can improve estimator accuracy without increasing the computational budget. In practice, underlying assumptions of the simulation model are subject to change, and new VA contracts are issued with different contract specifications. Simulation budgets are often scarce for new conditions. Chapter 4 uses transfer learning techniques for a quick adaptation of the neural network metamodel to new market conditions and new VA contracts. Our numerical experiments show that a judicious use of transfer learning can lead to substantially better metamodels than those trained from scratch. Chapter 5 concludes this thesis and discusses potential future research directions. Chapter 6 explores future applications of deep reinforcement learning in the dynamic hedging of VA contracts.

1.1 Risk Management of Variable Annuities with Nested Simulation

VA contracts are index-linked insurance products that offer policyholders the upside potential of equity markets while providing guarantees against downside risk. These products have gained a substantial amount of interest as they address the needs of individuals seeking both wealth accumulation and financial security, especially in the context of retirement planning. The key feature of VAs is their embedded guarantees, which ensure a specified minimum benefit regardless of market performance. Two VA contracts that are most relevant to this thesis are GMMB and GMWB. The GMMB guarantees that at the maturity of the contract, the policyholder will receive no less than the initial investment or a predetermined minimum amount. The GMWB allows policyholders to withdraw a specified percentage of their benefit base each period during the contract horizon (Hardy, 2003). While these guarantees make VAs more attractive to consumers by offering protection against market downturns and ensuring income stability, they also introduce significant financial risks for insurers. The embedded options within GMMB and GMWB expose insurers to market risk, longevity risk, and policyholder behavior risk. Therefore, an effective management of these risks is crucial for insurers to maintain solvency and meet regulatory capital requirements.

One of the primary risk management strategies employed by insurers to mitigate the financial risks associated with VAs is dynamic hedging. Dynamic hedging involves frequent adjustments to a portfolio of financial instruments to offset changes in guarantee values due to market movements (Hull and Basu, 2016). This strategy aims to neutralize the insurer's liability sensitivity to market fluctuations by constructing a hedging portfolio that replicates the guarantees' cash flows. However, dynamic hedging introduces complexity in estimating the insurer's overall risk exposure. One such example is the estimation of tail risk measures of a VA contract. It requires a stochastic modeling of the financial market and an accurate loss estimation of the hedging portfolio under different market scenarios. Nested simulation is a robust method for estimating risk measures in complex settings (Gordy and Juneja, 2010). In nested simulation, an outer simulation generates a set of market scenarios over the contract horizon, while an inner simulation estimates the contract loss in each scenario. By combining the results of the inner simulations across the outer scenarios, nested simulation provides an accurate estimate of the tail risk of the VA contract.

Despite its robustness, nested simulation is computationally intensive, often demanding significant computational resources and time. Each outer simulation scenario requires nu-

merous inner simulations to accurately estimate contract loss. This computational burden poses practical limitations, especially when a high degree of precision in tail risk estimation is required. To address this challenge, metamodeling techniques can be employed to approximate the inner simulation model and reduce the computational cost of nested simulation.

1.2 Metamodeling for Monte Carlo Simulation

Metamodeling, or surrogate modeling, approximates complex simulation models with simpler, computationally efficient models. In Monte Carlo simulations, metamodels reduce computational costs by approximating simulation outputs based on input variables (Kleijnen, 2018). This section introduces metamodeling for a Monte Carlo simulation, elaborates on its methodologies, and highlights its importance in practical applications.

Monte Carlo simulations are widely used for modeling and analyzing complex systems that are probabilistic in nature. These simulations often require a substantial amount of computational resources, especially when a high degree of accuracy or a high number of iterations is necessary (Glasserman, 2004). Metamodeling addresses this challenge by constructing an approximate model (known as a metamodel) that emulates the behavior of the original simulation model with much less computational effort. The metamodel serves as a predictive model that maps input variables to output responses. Learning from a set of simulation runs, the metamodel can generalize and predict outputs for new inputs without running the full simulation. Metamodeling is especially useful for computationally expensive simulation models. Replacing the simulation model with a metamodel significantly reduces computational costs. This allows for faster convergence of simulation-based estimators.

The process of metamodeling generally involves the following steps:

1. **Design of Experiments**: Select input combinations to run the original simulation and collect data.
2. **Building the Metamodel**: Use the collected data to train a metamodel that approximates the simulation model.
3. **Validation**: Assess the metamodel's accuracy and generalization capability.
4. **Application**: Use the metamodel to predict simulation outputs for new input combinations.

The development of advanced machine learning models and deep learning techniques has enhanced metamodeling approaches for Monte Carlo simulations, especially in high-dimensional and complex problem settings. Examples of machine learning applications include Jin et al. (2020), Tang et al. (2020), and Rosen et al. (2012). These methods offer powerful tools for capturing intricate patterns and nonlinear relationships that traditional metamodeling techniques might struggle to model effectively. In the following sections, we discuss machine learning algorithms in the context of metamodeling for Monte Carlo simulations and their applications in risk management.

1.3 Machine Learning for Risk Management Applications

Machine learning (ML) algorithms are essential tools for identifying complex patterns and relationships in data. In particular, they are well-suited for handling non-linearities and large-scale data structures, which are challenging for traditional statistical methods. In the context of risk management, ML algorithms have been widely used for predicting financial time series, estimating risk measures, and optimizing trading strategies. ML algorithms can be broadly categorized into supervised learning, unsupervised learning, and reinforcement learning. This section focuses on three widely used supervised learning models and two reinforcement learning algorithms that are most relevant to risk management applications.

1.3.1 Supervised Learning Models

Supervised learning is a fundamental approach in machine learning where the algorithm learns a mapping from inputs to outputs based on example input-output pairs Galton (1886). In this paradigm, a model is trained on a labeled dataset, which means that each training example is associated with an output label or value. The goal of a supervised learning paradigm is to learn a general rule that maps inputs (also known as features) to outputs (also known as targets), enabling the model to make accurate predictions on new, unseen data. Supervised learning is usually applied in two domains:

- **Classification**: The output variable is categorical, and the task is to assign inputs to one of several predefined categories to produce a qualitative prediction. Common uses in finance and actuarial applications are fraud detection and credit scoring.

- **Regression**: The output variable is continuous, and the task is to predict a real-valued number and produce a quantitative prediction. Examples in actuarial applications include predicting claim amounts, reserve estimates, asset pricing, and estimating tail risk measures of loss distributions.

This thesis focuses on regression models for risk management applications, where the goal is to predict a continuous target variable based on one or more input features. Given a dataset of n observations $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ is the d-dimensional input feature vector, and $y_i \in \mathbb{R}$ is the target variable, a supervised learning algorithm aims at learning a function $f(x; \theta) : \mathbb{R}^d \mapsto \mathbb{R}$ that to predict y from x using parameters θ.

Linear regression, kernel regression, and neural networks all fall under the supervised learning category and are widely used in risk management applications. Despite the differences in their functional forms, all supervised learning models can be evaluated on a similar set of metrics. The learning process, called training, minimizes a loss function $l(f(x; \theta), y)$ over the training data, which quantifies the difference between predicted and actual labels. A common loss function for regression problems is a quadratic loss function in the form of a MSE:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (f(x_i; \theta) - y_i)^2. \tag{1.1}$$

In most risk management applications, a critical task of users of supervised learning models is to design a suitable loss function that aligns with the objectives of the problem. Another important consideration is the choice of a suitable supervised learning algorithm. The choice may vary based on a trade-off among the complexity of the relationship, the interpretability of the model, and the computational resources available. In the following sections, we discuss four widely used supervised learning techniques in risk management applications: parametric regression, kernel regression, and two neural network architectures.

1.3.2 Parametric Regression Models

Regression models are the most common supervised learning algorithms used in risk management applications. A regression model predicts a continuous target variable based on one or more input features. A multiple linear regression is a simple and interpretable

model that attempts to predict a target variable as a linear combination of input features. It assumes a linear relationship between the target output and one or more input features. This method has been thoroughly explored in statistical literature. Bishop and Nasrabadi (2006) provide an extensive treatment of regression techniques in the broader context of machine learning. It is known as a parametric regression because it assumes a specific functional form for the input-output relationship with a finite set of parameters (Seber and Lee, 2012). A general form of a parametric regression model is given by:

$$f(x;\theta) = \beta_0 + \sum_{j=1}^{p} \beta_j \phi_j(x), \tag{1.2}$$

where p is a number of features, $\beta_0, \beta_1, \ldots, \beta_p$ are trainable regression coefficients, and $\phi_j(x)$ are basis functions that transform the input x to allow for non-linear modeling. Basis functions can be any functions of x that are chosen to capture the underlying structure of the data. Common choices include polynomial basis functions: $\phi_j(x) = x^j$ and Laguerre basis functions: $\phi_j(x) = e^{-x/2}L_j(x)$, where $L_j(x)$ are the Laguerre polynomials that are solutions to the Laguerre differential equation (Szeg, 1939). The training of parametric regression refers to the process of estimating the regression coefficients $\beta_0, \beta_1, \ldots, \beta_p$ that minimize the loss function. For a MSE loss metric, the optimal regression coefficients can be obtained by solving the normal equations. Parametric regression is powerful for modeling linear relationships when basis functions are known from expert knowledge or feature engineering (Hastie et al., 2009). These techniques utilize predefined functional forms to model the relationship between a set of independent variables and a dependent variable. However, when data exhibits complex, non-linear relationships, linear models fall short. While these methods are straightforward and interpretable, they impose strong assumptions about the underlying data structure. Often, expert knowledge is required to select the appropriate basis functions, which can limit the flexibility of the model. Extensions like polynomial regression and Generalized Linear Model (GLM) have been introduced to capture non-linearity in the data. Nevertheless, these models can still be limited in capturing highly complex patterns of the data.

1.3.3 Non-Parametric Regression Models

To overcome some of the previously mentioned limitations of parametric regression methods, non-parametric methods such as kernel regression (Hastie et al., 2009) are often employed. Kernel regression estimates the relationship by averaging the values of the target

variable over a local neighborhood of the input x. The kernel regression with the Nadaraya-Watson estimator is given by:

$$f(x;\theta) = \frac{\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right) y_i}{\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)}, \tag{1.3}$$

where $K(\cdot)$ is a kernel function, which assigns weights to the data points based on their distance to the input x, and h is a bandwidth parameter that controls the smoothness of the estimated function. The objective of this approach is to estimate a feature-label relationship directly from the data without imposing a specific functional form of the regression function. However, they come with several drawbacks that limit their effectiveness, especially in high-dimensional settings or for large datasets. One of the most important drawbacks of non-parametric regression methods is the curse of dimensionality Bellman (1966). As the number of features increases, the volume of the input space grows exponentially. Data points become sparse and the model may overfit to noise in the data. In addition, the computational cost associated with non-parametric methods often grows rapidly with the number of dimensions. Cross-validation and distance calculations are computationally expensive for non-parametric regression. These drawbacks render non-parametric regression impractical for high-dimensional, large-scale datasets.

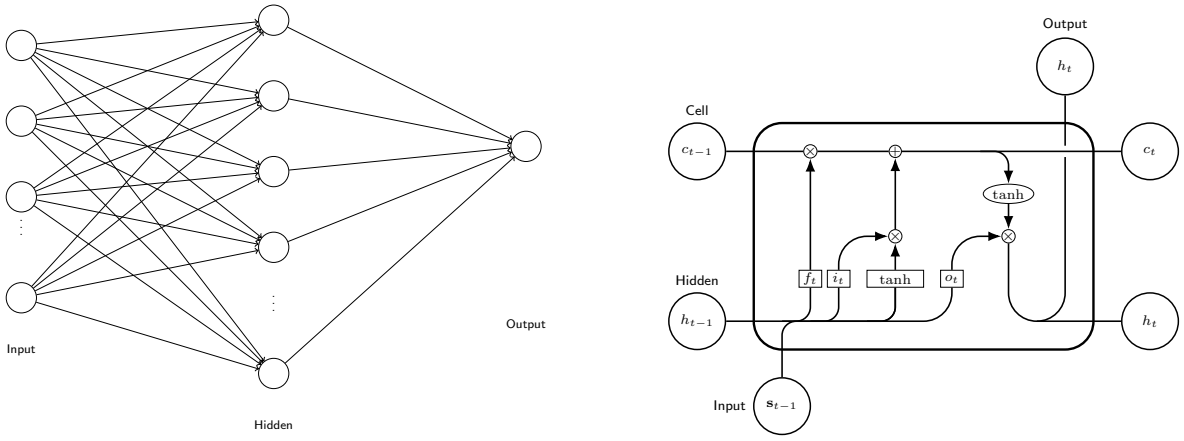
1.3.4 Feedforward Neural Networks

The progression from traditional parametric and non-parametric regression methods to neural network architectures has been driven by the need to model increasingly complex and high-dimensional data. Neural networks are powerful tools that overcome many limitations of traditional regression methods. They can learn complex, non-linear relationships in data without the need for explicit feature engineering. The most basic form of a neural network is the Feedforward Neural Network (FNN), which consists of an input layer, one or more hidden layers, and an output layer (Goodfellow et al., 2016a).

Figure 1.1a shows a FNN architecture with one hidden layer. The input layer receives the input features, which are then passed through the hidden layer(s) to the output layer. Each layer consists of multiple neurons, which apply a non-linear activation function to the weighted sum of the inputs.

$$f(x;\theta) = f^{(k)}(f^{(k-1)} \cdots (f^{(2)}(f^{(1)}(x;\theta^{(1)});\theta^{(2)}) \cdots ;\theta^{(k-1)});\theta^{(k)}), \tag{1.4}$$

where $\theta = (\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(k)})$ are trainable parameters of the neural network, $f^{(i)}$ is an i-th layer of the neural network, x is an input feature vector, $f^{(1)}(x;\theta^{(1)}) = x$, $f^{(k)}$ is an output



(a) FNN (b) Long Short-Term Memory (LSTM)

Figure 1.1: Neural Network Architectures

layer, k is a number of layers, and $f^{(i)}$ is an output of the i-th layer. To transition from layer $i-1$ to layer i,

$$f^{(i)}(z;\theta^{(i)}) = a^{(i)}(\beta_0^{(i)} + \sum_{j=1}^{p_i} \beta_j^{(i)} z), \tag{1.5}$$

where $a^{(i)}$ are nonlinear activation functions, $\theta^{(i)} = (\beta_0^{(i)}, \ldots, \beta_p^{(i)})$ are trainable parameters of the i-th layer, and p_i is a number of neurons in the previous layer. Equation (1.5) represents a single neuron in a neural network, which applies a linear transformation to the input followed by a non-linear activation function. The linear transformation is similar to the linear regression model in Equation (1.2), but the non-linear activation function allows the neural network to model complex, non-linear relationships in the data. The FNN is tuned by adjusting the trainable parameters θ while fixing the activation functions $a^{(i)}$. A typical choice for the activation function is the Rectified Linear Unit (ReLU), which is defined as $a(z) = \max(0, z)$ (Nair and Hinton, 2010). The activation function choice is crucial for training neural networks. While sigmoid and hyperbolic tangent functions were popular in early neural networks, ReLU has become the default activation function for deep networks due to its ability to mitigate the vanishing gradient problem and promote sparse activations (LeCun et al., 2015).

The key advantage of FNNs lies in their ability to perform automatic feature engineering without a direct human intervention. Unlike traditional machine learning models that

require manual selection and transformation of input features, neural networks learn to extract and compose features through their hidden layers during their training. Each hidden layer in the network captures higher-level abstractions of the input data by transforming the outputs of the previous layer through nonlinear activation functions (LeCun et al., 2015). Each layer builds upon the representations learned by the previous layer, enabling the network to capture multiple levels of abstraction. This characteristic is crucial for modeling complex datasets with intricate patterns and relationships (Bengio et al., 2013).

The last layer of the neural network, known as the output layer, typically performs a linear transformation of the features extracted by the preceding hidden layers. With $a^k(z) = z$, the last layer is equivalent to a multiple linear regression with its inputs as transformed, high-level features learned by the hidden layers. We can draw parallels between neural networks and traditional regression models. The main difference is that, in neural networks, the input features to the regression model are learned automatically rather than being manually specified.

The most well-known theorem in neural network theory is the universal approximation theorem (Hornik et al., 1989). It states that given appropriate activation functions a, a feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary accuracy. Despite this theoretical guarantee for single-layer neural networks, in practice, Deep Neural Network (DNN) with multiple hidden layers have been shown to be more effective at capturing complex patterns and relationships in data (LeCun et al., 2015). Examples include AlexNet (Krizhevsky et al., 2012), VGG (Simonyan and Zisserman, 2014), and ResNet (He et al., 2016), which have achieved state-of-the-art performance on image classification tasks. In this thesis, we focus on the application of deep neural networks, specifically LSTM, as metamodels for nested simulation procedures in risk management applications.

1.3.5 Long Short-Term Memory Networks

Building upon the capabilities of FNNs, we recognize that while FNNs are successful at capturing complex, nonlinear relationships through automatic feature learning, they are inherently limited when it comes to modeling sequential data or time-dependent patterns. This independence assumption limits their effectiveness in modeling financial time series, where temporal dependencies play a critical role. The stylized facts of financial time series, such as volatility clustering, fat tails, and autocorrelation, are challenging to capture with traditional FNNs due to the absence of memory in the model (Cont, 2001).

To overcome the limitations of FNNs in handling sequential input, Recurrent Neural Network (RNN)s were introduced. In an RNN, the hidden state at each time step is a function of both the current input and the hidden state from the previous time period (Elman, 1990):

$$h_t = f(x_t, h_{t-1}; \theta), \tag{1.6}$$

where h_t is the hidden state at time t, x_t is the input at time t, h_{t-1} is the hidden state at time $t-1$, and f is the recurrent function parameterized by θ. This architecture enables RNN to capture temporal dependencies by maintaining a dynamic internal state that reflects the memory of past inputs. However, traditional RNN suffer from the vanishing and exploding gradient problem, which hinders their ability to capture longer-term dependencies that often present in financial time series (Bengio et al., 1994). During training, gradients propagated backward through time can either diminish exponentially (known as vanishing gradients) or grow uncontrollably (known as exploding gradients). This limitation is particularly problematic in modeling long-term financial contracts and insurance guarantees, where patterns may span over extended periods.

To address these issues, a LSTM network was developed by Hochreiter and Schmidhuber (1997). It is a specialized form of RNNs designed to capture long-term dependencies more effectively with the help of RNN memory cells and gating mechanisms.

$$\begin{aligned}
i_t &= a(W_i x_t + U_i h_{t-1} + b_i), \\
f_t &= a(W_f x_t + U_f h_{t-1} + b_f), \\
o_t &= a(W_o x_t + U_o h_{t-1} + b_o), \\
g_t &= a(W_g x_t + U_g h_{t-1} + b_g), \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t, \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}$$

where $i_t, f_t, o_t, g_t, c_t, h_t$ are an input gate, a forget gate, an output gate, a cell input, a cell state, and a hidden state at time t, respectively. W_i, W_f, W_o, W_g, U_i, U_f, U_o, U_g are weight matrices, and b_i, b_f, b_o, b_g are bias vectors. a is the activation function, typically the sigmoid function, and $\odot$ denotes element-wise multiplication.

$$a(z) = \frac{1}{1 + e^{-z}}.$$

Figure 1.1b shows the architecture of an LSTM network. The gating mechanisms in LSTM networks effectively mitigate the vanishing and exploding gradient problem by regulating the flow of information through the network. The input gate i_t controls the flow of information into the cell state c_t, the forget gate f_t regulates the retention of information in the cell state, and the output gate o_t determines the information passed to the hidden state h_t. The cell input g_t is used to update the cell state based on the input x_t and the previous hidden state h_{t-1}. The cell state c_t acts as a memory unit that stores information over time, while the hidden state h_t captures the relevant information for the current time step. By incorporating memory cells and gating mechanisms, LSTM can effectively model long-term dependencies in sequential data in finance and actuarial applications.

The advancements in neural network optimization techniques, architectures, and training methodologies have significantly enhanced their usefulness in risk management applications. By effectively modeling complex, non-linear relationships and temporal dependencies, neural networks serve as powerful tools for addressing the computational challenges in estimating risk measures and developing effective risk mitigation strategies. This thesis aims at exploring the application and noise tolerance of LSTM networks in metamodeling for nested simulation procedures in risk management. In this thesis, we investigate the performance of LSTM networks in approximating the inner simulation model in a two-stage nested simulation procedure for index-linked insurance contracts. By leveraging the memory and sequential modeling capabilities of LSTM, we aim to improve the accuracy and efficiency of nested simulation procedures for risk management applications.

1.3.6 Training and Evaluation of Neural Networks in a Simulation Environment

The training of neural networks involves adjusting the trainable parameters of the network to minimize the loss function, which quantifies the error between the predictions and the labels. The training process aims at finding the optimal parameters that minimize this error. The most common approach is to use gradient descent methods with backpropagation, which is an efficient method for computing the gradients of the loss function with respect to the trainable parameters. Backpropagation computes the gradients by propagating the error derivatives backward through the network layers. This process allows the network to adjust the parameters in the direction of minimizing the loss function. The training process is typically performed over a number of epochs, where each epoch is one complete pass through the training dataset.

However, the training of neural networks is non-convex, and the optimization problem

is challenging. Stochastic Gradient Descent (SGD) is a popular optimization algorithm for training neural networks. It updates the parameters using a single or a small batch of training examples at a time, which makes it much faster than a crude gradient descent. However, SGD may oscillate and converge slowly, especially in non-convex problems (Goodfellow et al., 2016b). To address these issues, various adaptive learning rate methods have been proposed in the literature. The most popular approach is Adam optimizer, which is an SGD algorithm that adjusts the learning rate during training to improve convergence speed and stability (Kingma and Ba, 2014).

Another common technique to improve the training of neural networks is a regularization technique to prevent overfitting. One effective regularization method is dropout, which randomly sets a fraction of neurons to zero during each training iteration (Srivastava et al., 2014). This prevents neurons from co-adapting too much, encourages redundancy, and leads to a more robust model that generalizes better to unseen data. Early stopping is another regularization method that is relevant to our study. It involves monitoring the model's performance on a validation set and stopping training when the performance no longer improves (Prechelt, 2002). This helps prevent overfitting by stopping training before the model begins to fit to the noise in the data.

Evaluation of the performance of a neural network is challenging due to the absence of analytical tools. Existing machine learning literature addresses this challenge by splitting the data into three parts: training set, validation set, and test set. The training set is used to train the model, the validation set is used to tune the model hyperparameters, and the test set is used to evaluate how well the model generalizes to unseen data. The test set is not used during training and is only used for evaluation purposes.

Underfitting occurs when the training error is high, which means that the neural network fails to capture the underlying patterns in the training data. Overfitting occurs when the neural network fits to the noise in the training data and cannot generalize results to unseen data. In machine learning literature, overfitting is often quantified as the gap between the training error and the test error (Bishop and Nasrabadi, 2006). During training, as the test dataset is not used, this quantity is estimated by using the validation set. If the validation error is high compared to the training error, the model is likely overfitting to the training data.

In practice, the ultimate performance of a neural network is often evaluated based on error metrics on the test dataset and how it compares with the training error. The test error is often thought to be a reliable indicator of the model's generalization capability for most machine learning applications. However, real-world datasets often contain noise. As the test dataset is separated from the training dataset, it is often contaminated with noise.

Therefore, the test error may not be robust for evaluating the generalization capability of a neural network. In this thesis, we aim to evaluate the generalization capability of neural networks in a simulation environment. For a simulation model, we can generate a large number of datasets with different levels of noise. The noise level can be controlled by the design parameters of the simulation model, e.g., the number of replications. Instead of relying on the noisy test labels, we can generate a less noisy test labels using a higher number of replications, or even the true simulation model. In the rest of this thesis, we will use "test dataset" or "test labels" to refer to the noisy test dataset, and "true dataset", "true labels", or "true relationship" to refer to dataset generated from the true simulation model.

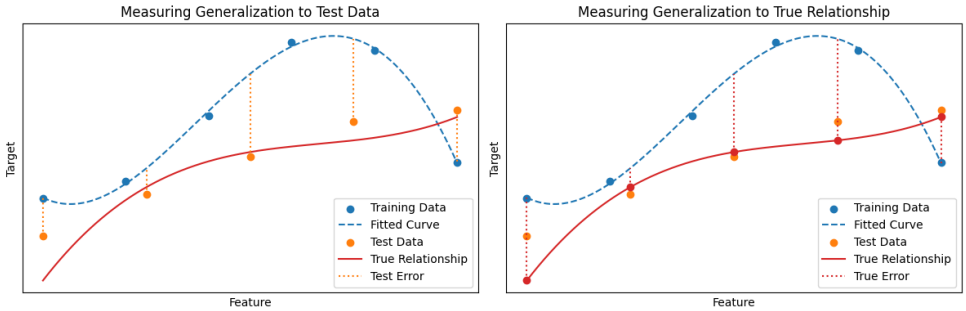


Figure 1.2: Different Levels of Noise

Figure 1.2 shows datasets with different levels of noise in a typical machine learning setting. The left panel illustrates the common practice of using a test dataset that is separate from the training dataset that is contaminated with noise. The right panel shows an alternative approach of using *true labels* generated from the true relationship. In a simulation context, the true relationship, which is the simulation model that generates the data, is visible to the user. Even if the true relationship is unknown, we can approximate it using a large number of replications and use it as the true labels. Comparing the errors of a neural network on the true labels can reveal its true generalization capability and robustness. Chapter 3 provides a detailed study of the generalization capability of LSTM networks in a simulation environment for risk management of VA contracts. Viewing the simulation environment as a data-generating process allows us to systematically study and evaluate the performance of neural networks within a controlled setting. This approach provides insights into their applicability in real-world risk management applications.

In the subsequent chapters, we dive deeper into these topics. Chapter 2 provides a comprehensive analysis of the convergence properties of existing nested simulation procedures

and their practical implications. Chapter 3 focuses on the development and evaluation of neural network metamodels, particularly LSTM networks, for enhancing the efficiency of nested simulations in risk management of VA contracts. We also examine the robustness and generalization capabilities of these models in the presence of simulation noise. Chapter 4 explores the use of transfer learning techniques to adapt neural network metamodels to new market conditions and contract specifications. Our approach enables rapid deployment in changing environments. Chapter 5 summarizes the key findings of this thesis and discusses future research directions that involve the robost use of neural networks in quantitative risk management applications. Chapter 6 explores future applications of transfer learning and deep reinforcement learning in the dynamic hedging of VA contracts.

Chapter 2

Nested Simulation Procedures in Financial Engineering: A Selected Review

2.1 Introduction

Nested simulation procedures are commonly used in financial engineering to estimate risk measures for portfolios of complex financial derivatives. The term *nested* refers to a nested estimation problem, in which the estimation of the risk measure requires two levels of Monte Carlo (MC) simulations. In a typical nested simulation procedure, an outer-level simulation model generates underlying risk factors, which are referred to as the *outer scenarios*. For each outer scenario, an inner-level simulation model generates scenario-wise samples of the portfolio losses, which are referred to as the *inner replications*.

The nested simulation procedure is computationally expensive due to its nested structure. Given a fixed computational budget, the nested simulation procedure should balance the number of outer scenarios and the number of inner replications to maximize computational efficiency. Gordy and Juneja (2010) first analyzed and proposed an optimal budget allocation for a standard nested simulation procedure. The term *standard* refers to using a standard MC estimator, the sample mean of the inner replications to estimate a scenario-wise portfolio loss for an outer scenario. Gordy and Juneja (2010) investigate an optimal budget allocation for a standard nested simulation procedure with respect to the MSE of the estimated risk measure.

The standard nested simulation procedure is computationally expensive with a wasteful use of the simulation budget, as only the inner replications from the same outer scenario are used in estimating the scenario-wise portfolio loss for that outer scenario. Subsequent research efforts have been made to improve the efficiency of nested simulation procedures by using inner replications from other outer scenarios. This is referred to as pooling. Different methods pool inner replications differently, using either a trained supervised learning metamodel or predefined likelihood ratio weights. Broadie et al. (2015) propose a regression-based nested simulation procedure, which uses a trained regression metamodel to estimate the scenario-wise portfolio loss for an outer scenario by pooling the inner replications from all outer scenarios. For certain risk measures, Broadie et al. (2015) show that it's optimal to allocate the entire simulation budget to outer-level simulations, keeping only 1 inner replication in each outer scenario. Similarly, Hong et al. (2017), Feng and Song (2020), and Zhang et al. (2022) use a kernel smoothing model, a likelihood ratio model, and a Kernel Ridge Regression (KRR) model as metamodels to pool the inner replications from all outer scenarios, respectively. In simulation studies, this approach—using a model of a simulation—is called metamodeling, and such models are termed metamodels (Barton, 1998). Another line of research is a Multi-Level Monte Carlo (MLMC) method analyzed in Giles and Haji-Ali (2019), which is a variance reduction technique that uses a hierarchy of approximations to the quantity of interest.

This paper presents a survey study of some popular nested simulation procedures that have both asymptotic convergence analysis and finite-sample performance evaluation. Many procedures have been proposed in the literature, but they are not directly comparable because they use different error metrics, different assumptions, and different numerical examples in their studies. Within a common analytical framework, we first summarize and compare the asymptotic convergence rates of these procedures. Their asymptotic convergence results are closely examined for their assumptions that guarantee the convergence. Furthermore, the existing literature has chosen different performance metrics to evaluate the asymptotic convergence of nested simulation procedures. In this chapter, we discuss the asymptotic convergence in terms of MSE of the estimator, and we provide a comprehensive comparison of their convergence rates and critical assumptions. We also discuss the connection between the MSE and the absolute error of the estimator in the context of nested simulation procedures.

In terms of finite-sample performance, different studies propose different examples in their numerical experiments, which introduces unfair advantages for certain simulation procedures over others. A fair comparison among popular methods is therefore urgently needed in the literature. Our numerical experiment in this chapter is the first of its kind to subject all the aforementioned simulation procedures to a fair comparison. As a result,

extensive numerical experiments are conducted to show, in practical examples, how well the finite-sample performance of a method matches its theoretical convergence behavior. Through our numerical examples, we compare the nested simulation procedures for different payoff complexities, problem dimensions, and risk measures. This chapter also provides a discussion of the computational complexity of different nested simulation procedures, including additional computational costs for training supervised learning metamodels and pooling inner replications.

The rest of this chapter is organized as follows. Section 2.2 introduces the nested simulation procedure and the standard MC estimator. Section 2.3 provides new theoretical results on the convergence of existing nested simulation procedures. Section 2.4 summarizes the asymptotic convergence orders and the critical assumptions of nested simulation procedures in the literature. Section 2.5 presents the numerical experiments and the comparisons of different nested simulation procedures with respect to different risk measures, problem dimensions, and payoff complexities. Section 2.6 discusses the computational complexity of different nested simulation procedures. Section 2.7 concludes this chapter.

2.2 Problem Formulation

In a nested estimation problem, we are interested in estimating the quantity

$$\rho(L) = \rho(L(X)),$$

where $X \in \mathcal{X} \subset \mathbb{R}^d$, and $L = L(X)$ is a random variable whose distribution is characterized by X. $L(X)$ is not subject to direct evaluation, but it is the output of

$$L(X) = \mathbb{E}\left[Y|X = x\right]|_{x=X}$$

Some common risk measures are in the nested expectation form as

$$\rho(L) = \mathbb{E}\left[h(L)\right]$$

where $h : \mathbb{R} \mapsto \mathbb{R}$ is a known nonlinear function. The nested expectation form is a general form that covers many risk measures of interest in financial engineering by varying the function h. Common forms of h in the literature include smooth functions, Lipschitz continuous functions, and indicator functions.

Definition 1 *A function $h : \mathbb{R} \mapsto \mathbb{R}$ is smooth if it is continuously differentiable up to the n-th order for some $n \in \mathbb{N}$, and its n-th and m-th derivatives are bounded for some $m \in \mathbb{N}$ and $m < n$.*

A quadratic tracking error with benchmark b, where $h(t) = (t-b)^2$, falls into this category. In the context of nested estimation for smooth h, different procedures require different versions of Definition 1 to specify the smoothness for their convergence analysis. Table 2.1 summarizes the smoothness assumption of h for different nested simulation procedures in the literature.

Nested Simulation Procedures	n	m
Regression	2	not required
Kernel smoothing	3	2
Kernel ridge regression	2	1
Likelihood ratio	2	1

Table 2.1: Smoothness assumption of h for different nested simulation procedures

The regression-based nested simulation procedure has the least stringent smoothness assumption. In general, a more complex metamodel requires a higher-order smoothness for h for its convergence analysis. Despite being more advanced than kernel smoothing, the Value at Risk (VaR)-based procedure requires a less stringent smoothness assumption on h. At first sight, it is counterintuitive that a more complex metamodel requires a less stringent smoothness assumption. This phenomenon is due to the fact that the convergence analysis for the VaR-based nested simulation procedure is shown in terms of an Absolute Error (AE), which is a less stringent error metric than MSE. Section 2.3.1 discusses the connection between the MSE and the AE of the estimator in the context of nested simulation procedures. In Table 2.1, the standard nested simulation procedure is the only procedure without an asymptotic convergence analysis for smooth h. Section 2.3.2 fills this gap by providing the asymptotic convergence analysis for the standard nested simulation procedure in terms of MSE.

Definition 2 *A function $h : \mathbb{R} \mapsto \mathbb{R}$ is Lipschitz continuous with a Lipschitz constant K if for all $t_1, t_2 \in \mathbb{R}$,*

$$|h(t_1) - h(t_2)| \leq K|t_1 - t_2|.$$

The mean excess loss over a threshold u is defined by setting $h(t) = \max\{t-u, 0\}$, which is a special case of a Lipschitz continuous function. The regression-based nested

simulation procedure is the only procedure that performs a theoretical analysis for Lipschitz continuous h. The other procedures only provide a convergence analysis for the special case of mean excess loss over a threshold u.

Definition 3 *A function $h : \mathbb{R} \mapsto \mathbb{R}$ is an indicator function if*

$$h(t) = \mathbb{I}_{\{t \geq u\}}$$

for some $u \in \mathbb{R}$.

The probability of a large loss over a threshold u is obtained by setting $h(t) = \mathbb{I}_{\{t \geq u\}}$.

Other risk measures of interest not in the nested expectation form include VaR and Conditional Value at Risk (CVaR) (Hardy and Saunders, 2022). The α-VaR of L is defined as

$$\text{VaR}_\alpha(L) = q_\alpha = \inf \{q : \Pr(L \leq q) \geq \alpha\}. \tag{2.1}$$

The α-CVaR of L is defined as

$$\text{CVaR}_\alpha(L) = \frac{1}{1-\alpha} \int_\alpha^1 q_v dv. \tag{2.2}$$

In the literature, CVaR are also known as Conditional Tail Expectation (CTE), Tail Value-at-Risk (TailVaR), and Expected Shortfall (ES).

The rest of this section reviews the existing nested simulation procedures in the literature. We unify their notations and provide a comprehensive comparison of how inner replications are generated and pooled for different procedures.

2.2.1 The Standard Nested Simulation Procedure

The standard nested simulation procedure first simulates M independent and identically distributed (i.i.d.) outer scenarios $X_1, \ldots, X_M$ from F_X, the distribution of X. For each X_i, again simulate Y_{ij}, $j = 1, \ldots, N$ from $F_{Y|X_i}$, the conditional distribution of Y given X_i. Given scenario i, the Y_{ij} are conditionally i.i.d. Let $\Gamma = M \cdot N$ denote the total simulation budget, $f_X(x)$ denote the density of X, and $\mathbf{X} = (X_1, \ldots, X_M)$ denote the vector of outer scenarios.

The standard nested simulation procedure estimates $L_i = L(X_i)$ with a standard MC estimator

$$\hat{L}_{N,i} = \frac{1}{N}\sum_{j=1}^{N} Y_{ij}, \quad Y_{ij} \sim F_{Y|X_i}.$$

Let $\hat{L}_{(i)}, \ldots, \hat{L}_{(M)}$ be the order statistics of $\hat{L}_{N,1}, \ldots, \hat{L}_{N,M}$. The standard nested simulation estimators for different forms of ρ are as follows:

1. Nested expectation form:

$$\hat{\rho}_{M,N} = \frac{1}{M}\sum_{i=1}^{M} h(\hat{L}_{N,i}) = \frac{1}{M}\sum_{i=1}^{M} h(\bar{Y}_{N,i}), \quad X_i \sim F_X.$$

2. Value at risk (VaR):

$$\hat{\rho}_{M,N} = \hat{L}_{(\lceil \alpha M \rceil)}.$$

3. CVaR:

$$\hat{\rho}_{M,N} = \hat{L}_{(\lceil \alpha M \rceil)} + \frac{1}{(1-\alpha)M}\sum_{i=1}^{M} \max\{\hat{L}_{N,i} - \hat{L}_{(\lceil \alpha M \rceil)}, 0\}. \tag{2.3}$$

Gordy and Juneja (2010) analyze the optimal budget allocation of the standard nested simulation procedure with respect to the MSE of the estimator $\hat{\rho}_{M,N}$ for hockey-stick h, indicator h, and VaR.

2.2.2 Multi-level Monte Carlo

The MLMC method is a variance reduction technique that uses a hierarchy of approximations to the quantity of interest, and it uses the difference between the approximations to reduce the variance of the estimator. The MLMC method is particularly useful when the quantity of interest is expensive to evaluate, and the standard MC estimator has a high variance. For the nested expectation form $\rho = \mathbb{E}[h(L)]$, the MLMC method selects an approximation level L with sufficient accuracy, and it uses the difference between the approximations to reduce the variance of the estimator. ~~The MLMC method estimates ρ with an MLMC estimator:~~

$$\hat{\rho}_{\Gamma}^{\text{MLMC}} = h(\hat{L}_{N_0}(X_{i,0})) + \sum_{\upsilon=1}^{\Upsilon} \frac{1}{M_\upsilon} \sum_{i=1}^{M_\upsilon} h(\hat{L}_{N_\upsilon}(X_{i,\upsilon})) - h(\hat{L}_{N_{\upsilon-1}}(X_{i,\upsilon-1})), \quad X_{i,\upsilon} \sim F_X,$$

where $X_{i,\nu}$ is the outer scenarios at level ν, Υ is a number of levels, M_υ is a number of outer scenarios at level υ, and N_υ is a number of inner replications at level υ. Applying the analysis of Giles (2015) in a nested simulation context, Giles and Haji-Ali (2019) show that the MLMC method can achieve a similar degree of accuracy as the standard nested simulation procedure with a lower total computational budget. The simulation budget Γ is a sum of the computational budget at each level, i.e., $\Gamma = \sum_{\upsilon=0}^{\Upsilon} M_\upsilon \cdot N_\upsilon$. Neither the standard nested simulation procedure nor the MLMC method involves pooling the inner replications from all outer scenarios. Pooling enhances sample efficiency and distinguishes advanced nested simulation procedures from the standard procedure.

2.2.3 Supervised Learning Metamodels

Supervised learning-based nested simulation procedures treat the inner simulation model $L(\cdot)$ as a black-box function, and they approximate $L(\cdot)$ with supervised learning metamodels. In this metamodeling approach, $L(\cdot)$ can be approximated by $\hat{L}^{\text{SL}}_{M,N}(\cdot)$, which is based on a family of chosen function and observations from the standard nested simulation procedure. Consider the observation pairs $(X_i, \hat{L}_{N,i})$ for $i \in \{1, \ldots, M\}$ as training data. We can use supervised learning to approximate L_i by $\hat{L}^{\text{SL}}_{M,N}(X_i)$ and to pool the inner replications from all outer scenarios. Specifically, $\hat{L}^{\text{SL}}M, N(\cdot)$ denotes a supervised learning model trained on $(X_i, \hat{L}N, i)$ for $i \in 1, \ldots, M$. This is the output of the standard nested simulation procedure with M outer scenarios and N inner replications. To evaluate the convergence of the supervised learning-based nested simulation procedure, we consider the convergence behavior of the supervised learning model $h(\hat{L}^{\text{SL}}_{M,N}(\cdot))$ to the true risk measure $\rho = h(L(\cdot))$. More specifically, we want to study the following quantity

$$\mathbb{E}\left[h(\hat{L}^{\text{SL}}_{M,N}(X))\right] - \rho, \quad X \sim F_X. \tag{2.4}$$

However, studying the expectation in Equation (2.4) is unrealistic, as in practice the risk measure ρ is estimated with a finite number of samples. Using the M *training* samples, a supervised learning-based nested MC estimator of ρ is given by

1. Nested expectation form:

$$\hat{\rho}^{\text{SL,Train}}_{M,N} = \frac{1}{M}\sum_{i=1}^{M} h(\hat{L}^{\text{SL}}_{M,N}(X_i)), \quad X_i \sim F_X. \tag{2.5}$$

2. VaR:

$$\hat{\rho}^{\text{SL,Train}}_{M,N} = \hat{L}^{\text{SL}}_{(\lceil \alpha M \rceil)}.$$

3. VaR:

$$\hat{\rho}^{\text{SL,Train}}_{M,N} = \hat{L}^{\text{SL}}_{(\lceil \alpha M \rceil)}.$$

4. CVaR:

$$\hat{\rho}^{\text{SL,Train}}_{M,N} = \hat{L}^{\text{SL}}_{(\lceil \alpha M \rceil)} + \frac{1}{(1-\alpha)M}\sum_{i=1}^{M} \max\{\hat{L}^{\text{SL}}_{M,N}(X_i) - \hat{L}^{\text{SL}}_{(\lceil \alpha M \rceil)}, 0\}.$$

where $\hat{L}^{\text{SL}}_{(\lceil \alpha M \rceil)}$ is the $\lceil \alpha M \rceil$-th order statistic of $\hat{L}^{\text{SL}}_{M,N}(X_1), \ldots, \hat{L}^{\text{SL}}_{M,N}(X_M)$.

After a supervised learning metamodel is trained, it can be used to make predictions for all $X \in \mathcal{X}$. Instead of using the same training samples to estimate the risk measure, we can use metamodel predictions for a new set of M' *test* samples of X, namely $\tilde{\mathbf{X}} = \{\tilde{X}_1, \ldots, \tilde{X}_{M'}\}$. The resulting supervised learning-based estimator of $\rho^{\text{SL,Test}}$ is given by

1. Nested expectation form:

$$\hat{\rho}^{\text{SL,Test}}_{M,N,M'} = \frac{1}{M'}\sum_{i=1}^{M'} h(\hat{L}^{\text{SL}}_{M,N}(\tilde{X}_i)), \quad \tilde{X}_i \sim F_X. \tag{2.6}$$

2. VaR:

$$\hat{\rho}^{\text{SL,Test}}_{M,N,M'} = \hat{L}^{\text{SL}}_{(\lceil \alpha M' \rceil)}.$$

3. VaR:

$$\hat{\rho}^{\text{SL,Test}}_{M,N,M'} = \hat{L}^{\text{SL}}_{(\lceil \alpha M' \rceil)}.$$

4. CVaR:

$$\hat{\rho}^{\text{SL,Test}}_{M,N,M'} = \hat{L}^{\text{SL}}_{(\lceil \alpha M' \rceil)} + \frac{1}{(1-\alpha)M'}\sum_{i=1}^{M'} \max\{\hat{L}^{\text{SL}}_{M,N}(\tilde{X}_i) - \hat{L}^{\text{SL}}_{(\lceil \alpha M' \rceil)}, 0\},$$

where $\hat{L}^{\mathrm{SL}}_{(\lceil \alpha M' \rceil)}$ is an $\lceil \alpha M' \rceil$-th order statistic of $\hat{L}^{\mathrm{SL}}_{M,N}(\tilde{X}_1), \ldots, \hat{L}^{\mathrm{SL}}_{M,N}(\tilde{X}_{M'})$. Note that $\hat{L}^{\mathrm{SL}}_{M,N}(\cdot)$ represents the predictions of a supervised learning model trained on the samples $(X_1, \hat{L}_{N,1}), \ldots, (X_M, \hat{L}_{N,M})$.

Existing literature on nested simulation procedures has proposed different supervised learning metamodels to approximate the true function $L(\cdot)$. In this study, we focus on the metamodeling approaches that include theoretical convergence results. Methods that include theoretical convergence results are regression (Broadie et al., 2015), kernel smoothing (Hong et al., 2017), and KRR (Wang et al., 2022). Their estimators of $L(\cdot)$ are given by $\hat{L}^{\mathrm{REG}}_{M,N}(\cdot)$, $\hat{L}^{\mathrm{KS}}_{M,N}(\cdot)$, and $\hat{L}^{\mathrm{KRR}}_{M,N}(\cdot)$, respectively.

- Regression:
$$\hat{L}^{\mathrm{REG}}_{M,N}(X) = \Phi(X)\hat{\beta},$$
where Φ is a chosen basis, and $\hat{\beta}$ is estimated from the training samples. In Broadie et al. (2015), the convergence analysis is performed on the expectation (Equation (2.4)) and for smooth and Lipschitz continuous h.

- Kernel smoothing:
$$\hat{L}^{\mathrm{KS}}_{M,N}(X) = \frac{\sum_{i=1}^{M} \hat{L}_{N,i} K_w(X - X_i)}{\sum_{i=1}^{M} K_w(X - X_i)},$$
where K_w is a kernel function with bandwidth w. Nadaraya (1964) and Watson (1964) originally proposed this kernel smoothing method for a non-parametric regression, and it is widely known in the literature as a Nadaraya-Watson kernel regression. Hong et al. (2017) provides convergence analysis for metamodel predictions of the training samples (Equation (2.5)), and the convergence analysis is provided for smooth, hockey-stick, and indicator h.

- K-Nearest Neighbors (kNN) (Mack, 1981):
$$\hat{L}^{\mathrm{kNN}}_{M,N}(x) = \frac{\frac{1}{MR_M^d} \sum_{i=1}^{M} \hat{L}_{N,i} K\left(\frac{x - X_i}{R_M}\right)}{\frac{1}{MR_M^d} \sum_{i=1}^{M} K\left(\frac{x - X_i}{R_M}\right)},$$
where R_k is a Euclidean distance between X_i and its k-th nearest neighbor, and $K : \mathbb{R}^d \mapsto \mathbb{R}$ is a bounded, nonnegative kernel function satisfying
$$\int K(u)du = 1,$$
$$K(u) = 0, \quad \text{for } \|u\| \geq 1,$$

where $\|\cdot\|$ is the Euclidean norm, and k is a number of nearest neighbors to consider. The kNN method is implemented as a metamodel for numerical experiments in Hong et al. (2017), but the convergence analysis is not provided.

- VaR:

$$\hat{L}^{\mathrm{KRR}}_{M,N}(X) = \underset{g\in\mathcal{N}_\Psi(\Omega)}{\arg\min}\left(\frac{1}{M}\sum_{i=1}^{M}(\hat{L}_{N,i}-L_i)^2+\lambda\|L\|^2_{\mathcal{N}_\Psi(\Omega)}\right),$$

where $\mathcal{N}_\Psi(\Omega)$ is a Reproducing Kernel Hilbert Space (RKHS) with a kernel Ψ defined on a domain Ω, and λ is a regularization parameter as in a ridge regression. More specifically, Φ is a Matérn kernel with a smoothness parameter ν and a length scale parameter υ. Wang et al. (2022) provides a convergence analysis for the AE (Equation (2.8)), and the convergence analysis is provided for a smooth, a Lipschitz continuous, and an indicator h, VaR, and CVaR.

These supervised learning metamodels approximate the inner simulation model $L(\cdot)$ with functions in the form of $\hat{L}^{\mathrm{SL}}_{M,N}(\cdot)$, and the metamodels are used to pool the inner replications from all outer scenarios to estimate the risk measure ρ. In the simulation literature, using models to approximate simulation models is often called metamodeling; such models are referred to as metamodels. Supervised learning models are broadly classified into regression and classification models depending on whether the target variable is continuous or discrete. In our context, the target variable L is continuous, and the supervised learning models refer to regression models. Regression models can be further categorized into parametric and non-parametric regressions. Among the supervised learning models we consider, multiple linear regression is parametric, while kernel smoothing and KRR are non-parametric.

We aim to minimize the MSE of the supervised learning-based nested simulation estimators $\hat{\rho}^{\mathrm{SL,Train}}_{M,N}$ and $\hat{\rho}^{\mathrm{SL,Test}}_{M,N,M'}$ subject to the total simulation budget Γ. We are also interested in the order of convergence of the estimator for all nested simulation procedures as the total simulation budget Γ approaches infinity.

$$\begin{aligned}\min_{M,N}\quad & \mathrm{MSE}(\hat{\rho}^{\mathrm{SL}}_{M,N}) = \mathbb{E}\left[\left(\hat{\rho}^{\mathrm{SL}}_{M,N}-\rho\right)^2\right],\\ \text{subject to}\quad & M\cdot N=\Gamma.\end{aligned} \tag{2.7}$$

For the more complex metamodels, the convergence analysis of their MSEs is more challenging. More specifically, the VaR-based nested simulation (Wang et al., 2022) provides

a convergence analysis in terms of AE.

$$\begin{aligned} \min_{M,N} \quad & \mathrm{AE}(\hat{\rho}^{\mathrm{SL}}_{M,N}) = \mathbb{E}\left[\left|\hat{\rho}^{\mathrm{KRR}}_{M,N} - \rho\right|\right] \\ \text{subject to} \quad & M \cdot N = \Gamma \end{aligned} \tag{2.8}$$

Section 2.3.1 draws the connection between the MSE and the AE of the estimator in the context of nested simulation procedures.

2.2.4 Likelihood Ratio Method

Instead of using a supervised learning model as a metamodel, Zhang et al. (2022) uses the likelihood ratio weights to pool the inner replications from all outer scenarios. Here, we restrict our attention to problems in the nested expectation form which outer scenarios characterize the stochasticity of the inner simulation model. Specifically,

$$Y = Y(H, X),$$

where H is a random variable whose distribution is specified by the outer scenarios X. We denote the conditional distribution of $H|X$ by $f_{H|X}$. For a specific scenario X_i, we write $f_{H|X}(\cdot|X_i)$. To reconcile with previously established notations, we note that inner simulation outputs Y_{ij} can be written as

$$Y_{ij} = Y(H_{ij}, X_i),$$

where $H_{ij} \sim f_{H|X}(\cdot|X_i)$. Suppose that one can generate random variable H from some sampling distribution f_H. Then, the likelihood ratio estimator of ρ is given by

$$\hat{\rho}^{\mathrm{LR}}_{M,N} = \frac{1}{M}\sum_{i=1}^{M} h(\hat{L}^{\mathrm{LR}}_{N,i}),$$

where the inner replications are pooled by the likelihood ratio weights with

$$\hat{L}^{\mathrm{LR}}_{N,i} = \frac{1}{N}\sum_{j=1}^{N} Y(H_j, X_i)\frac{f_{H|X}(H_j|X_i)}{f_H(H_j)}, \quad H_j \sim f_H, \quad i = 1, \ldots, M.$$

In Section 2.5.4, we implement the likelihood ratio method and share its fast convergence in finite-sample experiments despite the costly computation of the likelihood ratio weights (Section 2.6).

2.2.5 Problem Statement

With a total simulation budget Γ, we are interested in the orders of convergence of estimators for all nested simulation procedures, which are measured by their MSE about the risk measure ρ.

$$\begin{aligned} \min \quad & \mathbb{E}\left[(\hat{\rho}_{M,N} - \rho)^2\right], \\ \text{subject to} \quad & M \cdot N = \Gamma, \end{aligned} \tag{2.9}$$

where Γ is a simulation budget for a nested simulation procedure. A special case is the MLMC method, where the total simulation budget Γ is the sum of the simulation budgets at all levels.

2.3 Asymptotic Analysis

In the first part of this section, we summarize the existing asymptotic convergence results of the nested simulation procedures in the literature, and we compare their critical assumptions that guarantee the convergence. The remainder of this section aims to fill the gap in the literature by offering the following steps:

- provide the asymptotic convergence results for the standard nested simulation procedure,
- show the connections between the convergence in MSE and the convergence in probabilistic order for AE in the context of nested simulation procedures, and
- illustrate the asymptotic rate of convergence of a kNN-based nested simulation procedure.

Table 2.2 summarizes the existing asymptotic convergence results of nested simulation procedures for MSE in the literature. A ✓ indicates there exists an asymptotic convergence result for the corresponding estimator, a $\times$ indicates there does not exist an asymptotic convergence result, a $\star$ indicates an asymptotic result is not available in the literature but is provided in this study, and a $\diamondsuit$ indicates that an asymptotic convergence result exists in the literature, though in a weaker form. Gordy and Juneja (2010) provide asymptotic convergence results for the standard nested simulation procedure with a hockey-stick h,

Estimator for L	**Smooth h**	**Lipschitz(hockey-stick h)**	**Indicator h**	**VaR**	**CVaR**
Standard MC	$\star$	$\star(\checkmark)$	$\checkmark$	$\checkmark$	$\times$
Multi-level MC	$\times$	$\times(\times)$	$\checkmark$	$\times$	$\times$
Regression	$\checkmark$	$\checkmark(\checkmark)$	$\times$	$\times$	$\times$
Kernel smoothing	$\checkmark$	$\times(\checkmark)$	$\checkmark$	$\times$	$\times$
Kernel ridge regression	$\Diamond$	$\times(\Diamond)$	$\Diamond$	$\Diamond$	$\Diamond$
Likelihood ratio	$\checkmark$	$\times(\checkmark)$	$\checkmark$	$\times$	$\times$

Table 2.2: Existing asymptotic convergence results of nested simulation procedures for MSE

an indicator h and VaR. Their CVaR analysis is incomplete as the VaR is assumed to be known but not estimated in the convergence proof. When the VaR is known, the CVaR analysis reduces to the nested expectation form with h being a hockey-stick function. Giles and Haji-Ali (2019) provide asymptotic convergence results for the MLMC method with an indicator h. Broadie et al. (2015) provide asymptotic convergence results for the regression-based nested simulation procedure with a smooth h and a Lipschitz continuous h. The Lipschitz continuous family includes the hockey-stick function as a special case, thus the convergence result for the hockey-stick function is implied. Hong et al. (2017) and Zhang et al. (2022) provide asymptotic convergence results with the nested expectation form for the kernel smoothing-based procedure and likelihood ratio-based nested simulation procedure, respectively. The convergence results for a Lipschitz continuous h cannot be directly inferred from the analysis for a hockey-stick function.

While most of the literature focuses on the MSE of the estimator of ρ, Wang et al. (2022) analyze the asymptotic convergence of the estimator of ρ in terms of AE. Let $\hat{\rho}$ be the estimator of ρ. Then the AE of $\hat{\rho}$ about ρ is defined as

$$\mathrm{AE}\left(\hat{\rho}\right) = \left|\hat{\rho} - \rho\right|.$$

In Wang et al. (2022), the authors of the VaR-based nested simulation procedures claim to have bridged the gap between the cubic and square root convergence rates of nested simulation procedures. However, they analyze convergence in probabilistic order, and it is only applicable in terms of AE. Instead of showing the convergence of the VaR-based estimator in terms of MSE as in Gordy and Juneja (2010), we show the connections between the convergence in MSE and the convergence in probabilistic order for AE. Our findings in Section 2.3.1 show that the analysis of Wang et al. (2022) indeed bridges the gap, but only in terms of convergence in probabilistic order for AE.

2.3.1 Connections between Convergence in MSE and AE

This section establishes the connections between the convergence in MSE and the convergence in probabilistic order for AE in the context of nested simulation procedures. In order to show the connections between the convergence in MSE and the convergence in probabilistic order for AE, we first need to state the definition for a sequence of random variables to converge in those two forms.

Definition 4 *Let $\hat{\rho}_\Gamma$ be an estimator of ρ with a simulation budget of Γ. We write $\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right] = \mathcal{O}\left(\Gamma^{-\xi}\right)$, that is, $\hat{\rho}_\Gamma$ converges in MSE to ρ in order ξ if there exists a constant C such that*

$$\limsup_{\Gamma\to\infty} \frac{\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right]}{\Gamma^{-\xi}} \leq C.$$

Definition 5 *Let $\hat{\rho}_\Gamma$ be an estimator of ρ with a simulation budget of Γ. We write $|\hat{\rho}_\Gamma - \rho| = \mathcal{O}_{\mathbb{P}}(\Gamma^{-\xi})$, that is $\hat{\rho}_\Gamma$ converges in probabilistic order ξ to ρ if for a sufficiently large Γ, for any $\epsilon > 0$ there exists a C such that*

$$\mathbb{P}\left(|\hat{\rho}_\Gamma - \rho| \geq C\Gamma^{-\xi}\right) \leq \epsilon.$$

We start our analysis by showing the convergence in probabilistic order from the convergence in MSE. Let $\hat{\rho}_\Gamma$ be an estimator of ρ with a simulation budget of Γ, and assume that $\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right] = \mathcal{O}\left(\Gamma^{-\xi-\delta}\right)$ for an arbitrarily small $\delta > 0$. Then, from the definition of convergence in MSE,

$$\limsup_{\Gamma\to\infty} \frac{\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right]}{\Gamma^{-\xi-\delta}} \leq C.$$

for a constant $C > 0$.

The above inequality implies that

$$\limsup_{\Gamma\to\infty} \frac{\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right]}{\Gamma^{-\xi}} = 0.$$

Therefore, by Chebyshev's inequality, for any $\epsilon > 0$, we have

$$\mathbb{P}\left(|\hat{\rho}_\Gamma - \rho|\,\Gamma^{\xi/2} > C\right) \leq \frac{\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\,\Gamma^{\xi}\right]}{C^2} \to 0$$

as $\Gamma \to \infty$.

Therefore, $\hat{\rho}_\Gamma$ converges in probabilistic order $\xi/2$ to ρ as $\Gamma \to \infty$, that is,

$$|\hat{\rho}_\Gamma - \rho| = \mathcal{O}_{\mathbb{P}}\left(\Gamma^{-\xi/2}\right).$$

2.3.2 Asymptotic Analysis for the Standard Nested Simulation Procedure

In Gordy and Juneja (2010), the authors analyze the asymptotic convergence of the standard nested simulation procedure in terms of MSE. The analysis is complete for the nested expectation form where h is either an indicator function or a hockey-stick function and VaR. For the nested expectation form where h is a smooth function or a Lipschitz continuous function, the analysis is incomplete. In this section, we fill in the holes for the analysis of Gordy and Juneja (2010).

Assumption 1 *$h(L)$ has finite second moment, i.e., $\mathbb{E}\left[(h(L))^2\right] < \infty$.*

Assumption 1 is a standard assumption in the literature (Hong et al., 2017) for the analysis of nested simulation procedures to analyze the convergence of the variance of $\hat{\rho}_{M,N}$.

Assumption 2 *$\hat{L}_N(X) = L(X) + \bar{Z}_N(X)$, where the simulation noise $\bar{Z}_N(X)$ has zero mean and variance $\nu(X)/N$, and the conditional variance $\nu(X)$ is bounded, i.e.,*

$$\sup_{x \in \mathcal{X}} \nu(x) \leq C_{\nu,1} < \infty.$$

For simplicity, we abbreviate $\hat{L}_N(X)$ as $\hat{L}_N$, $\bar{Z}_N(X)$ as $\bar{Z}_N$, and $\bar{Z}_{N,i}$ as $\bar{Z}_{N,i}$. Let $\rho_M = \frac{1}{M}\sum_{i=1}^{M} h(L_i)$ be a nested MC estimator with the true function g. The MSE of the standard nested simulation procedure can be decomposed into two ways.

$$
\begin{aligned}
&\mathbb{E}\left[(\hat{\rho}_{M,N}-\rho)^2\right] \\
&\leq 2\mathbb{E}\left[(\hat{\rho}_{M,N}-\rho_M)^2\right]+2\mathbb{E}\left[(\rho_M-\rho)^2\right] \\
&= 2\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M}h\left(\hat{L}_{N,i}\right)-\frac{1}{M}\sum_{i=1}^{M}h\left(L_i\right)\right)^2\right] \\
&\quad + 2\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M}h\left(L_i\right)-\mathbb{E}\left[h(L)\right]\right)^2\right] \\
&= 2\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M}h\left(\hat{L}_{N,i}\right)-h\left(L_i\right)\right)^2\right]+\frac{2}{M}\mathrm{Var}(h(L)) \\
&= 2\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M}h\left(\hat{L}_{N,i}\right)-h\left(L_i\right)\right)^2\right]+\mathcal{O}(M^{-1}). \qquad (2.10)
\end{aligned}
$$

In Equation (2.10), the first inequality follows from Cauchy-Schwarz inequality. One way to express the Cauchy-Schwarz inequality is $(x_1y_1+x_2y_2)^2 \leq (x_1^2+x_2^2)(y_1^2+y_2^2)$. Let $\mathbf{x}=(x_1,x_2)$ and $\mathbf{y}=(y_1,y_2)$. The Cauchy-Schwarz inequality can be written as

$$
\langle \mathbf{x},\mathbf{y}\rangle^2 = (\|\mathbf{x}\|\cdot\|\mathbf{y}\|\cdot\cos a)^2 \leq \|\mathbf{x}\|^2\cdot\|\mathbf{y}\|^2.
$$

By setting $x_1=\rho_{M,N}-\rho_M$, $x_2=\rho_M-\rho$, $y_1=y_2=1$, we have

$$
\begin{aligned}
(\rho_{M,N}-\rho_M+\rho_M-\rho)^2 &\leq \left((\rho_{M,N}-\rho_M)^2+(\rho_M-\rho)^2\right)\left(1^2+1^2\right) \\
&= 2\left((\rho_{M,N}-\rho_M)^2+(\rho_M-\rho)^2\right) \\
&= 2(\rho_{M,N}-\rho_M)^2+2(\rho_M-\rho)^2.
\end{aligned}
$$

The last equality follows from Assumption 1. The analysis of the first term is different for smooth and Lipschitz continuous h. We will analyze them separately below.

Another way to decompose the MSE is a bias-variance decomposition as in Gordy and Juneja (2010).

$$
\begin{aligned}
\mathbb{E}\left[(\hat{\rho}_{M,N}-\rho)^2\right] &= \mathbb{E}\left[(\hat{\rho}_{M,N}-\mathbb{E}\left[\hat{\rho}_{M,N}\right])^2\right]+\left(\mathbb{E}\left[\hat{\rho}_{M,N}\right]-\rho\right)^2 \\
&= \mathrm{Var}(\hat{\rho}_{M,N})+\mathrm{Bias}^2(\hat{\rho}_{M,N}), \qquad (2.11)
\end{aligned}
$$

where $\mathrm{Var}(\hat{\rho}_{M,N})$ is variance of the estimator and $\mathrm{Bias}(\hat{\rho}_{M,N})$ is bias of the estimator about ρ. For a smooth function h, both ways of decomposing the MSE lead to the same order of convergence. However, for a Lipschitz continuous function h, Equation (2.11) leads to a stronger order of convergence than Equation (2.10). In the following, we analyze the convergence of the standard nested simulation procedure for a smooth and a Lipschitz continuous h by using both decomposition methods. Their critical assumptions and convergence results are summarized and compared.

A Smooth Function h

Assumption 3 *The function h has bounded first and second order derivative, i.e.,*

$$\sup_{x\in\mathcal{X}} |h'(x)| \leq C_{h,1} < \infty,$$
$$\sup_{x\in\mathcal{X}} |h''(x)| \leq C_{h,2} < \infty.$$

Assumption 3 is similar to the smoothness assumption in Wang et al. (2022). Since h is a smooth function, Taylor expansion can be applied to the first term in Equation (2.10).

$$\begin{aligned}
&\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h\left(\hat{L}_{N,i}\right) - h\left(L_i\right)\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h'\left(L_i\right)\left(\hat{L}_{N,i} - L_i\right) + \frac{1}{2M}\sum_{i=1}^{M} h''\left(z_i\right)\left(\hat{L}_{N,i} - L_i\right)^2\right)^2\right] \\
&\leq 2\,\underbrace{\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h'\left(L_i\right)\left(\hat{L}_{N,i} - L_i\right)\right)^2\right]}_{S_1} \\
&\quad + 2\,\underbrace{\mathbb{E}\left[\left(\frac{1}{2M}\sum_{i=1}^{M} h''\left(z_i\right)\left(\hat{L}_{N,i} - L_i\right)^2\right)^2\right]}_{S_2},
\end{aligned} \tag{2.12}$$

where z_i is between L_i and $\hat{L}_{N,i}$, and the last inequality is due to $2ab \leq a^2 + b^2$ for any $a, b \in \mathbb{R}$. The two terms on the right-hand side of Equation (2.12) are analyzed separately. We start with the first term S_1.

$$
\begin{aligned}
S_1 &= \mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h'\left(L_i\right)\left(\hat{L}_{N,i} - L_i\right)\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h'\left(L_i\right)\bar{Z}_{N,i}\right)^2\right] \\
&\leq C_1^2\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} \bar{Z}_{N,i}\right)^2\right] \\
&= C_1^2\mathbb{E}\left[\frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} \bar{Z}_{N,i}\bar{Z}_{N,j}\right] \\
&= C_1^2\mathbb{E}\left[\frac{1}{M^2}\sum_{i=1}^{M} \bar{Z}_{N,i}^2 + \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j\neq i}^{M} \bar{Z}_{N,i}\bar{Z}_{N,j}\right] \\
&\leq \frac{C_1^2 C_{\nu,1}}{MN} = \mathcal{O}(M^{-1}N^{-1}),
\end{aligned} \tag{2.13}
$$

where the last inequality in Equation (2.13) is due to Assumption 2, independence of X_i and X_j for $i \neq j$, and the fact that the inner simulation noise has zero mean, i.e., $\mathbb{E}\left[\bar{Z}_N\right] = 0$. It remains to analyze the second term S_2, where Assumption 4 is necessary to ensure the existence of the fourth moment of the simulation noise and the convergence of the second term.

Assumption 4 *The fourth moment of simulation noise $\bar{Z}_N(X)$ follows $\mathbb{E}\left[\left(\bar{Z}_N(X)\right)^4\right] = \nu_2(X)/N^2$, where $\nu_2(X)$ is bounded, i.e., there exists $C_{\nu,2} > 0$ such that $\nu_2(X) \leq C_{\nu,2}$ for all $x \in \mathbb{R}$.*

$$
\begin{aligned}
S_2 &= \mathbb{E}\left[\left(\frac{1}{2M}\sum_{i=1}^{M} h''(z_i)\left(\hat{L}^{\mathrm{SL}}_{M,N}(X_i) - L_i\right)^2\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{2M}\sum_{i=1}^{M} h''(z_i)\,\bar{Z}^2_{N,i}\right)^2\right] \\
&\leq C_2^2\mathbb{E}\left[\left(\frac{1}{2M}\sum_{i=1}^{M} \bar{Z}^2_{N,i}\right)^2\right] \\
&= C_2^2\mathbb{E}\left[\frac{1}{2M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} \bar{Z}^2_{N,i}\bar{Z}^2_{N,j}\right] \\
&= C_2^2\mathbb{E}\left[\frac{1}{2M^2}\sum_{i=1}^{M} \bar{Z}^4_{N,i} + \frac{1}{2M^2}\sum_{i=1}^{M}\sum_{j\neq i}^{M} \bar{Z}^2_{N}(X_i)\bar{Z}^2_{N}(X_j)\right] \\
&\leq C_2^2\left(\frac{C_{\nu,2}M}{2M^2N^2} + \frac{C^2_{\nu,1}M(M-1)}{2M^2N^2}\right) = \mathcal{O}(N^{-2}),
\end{aligned} \tag{2.14}
$$

where the second inequality is due to Assumption 3, and the last inequality follows from Assumption 4 and the fact that $\hat{L}_N$ is a standard MC estimator of L. Combining Equation (2.10), 2.12, 2.13, and 2.14, we have

$$
\mathbb{E}\left[(\hat{\rho}_{M,N} - \rho)^2\right] = \mathcal{O}(M^{-1}) + \mathcal{O}(N^{-2}). \tag{2.15}
$$

Setting $M = \mathcal{O}(\Gamma^{2/3})$ and $N = \mathcal{O}(\Gamma^{1/3})$, we provide the same rate of convergence as obtained for other risk measures in Gordy and Juneja (2010). However, we observe that Equation (2.13) converges in the order of $\mathcal{O}(M^{-1}N^{-1})$. The cross term here is potentially problematic, as setting N as a negative power of M may result in Equation (2.13) to be divergent.

In order to avoid the cross term, we proceed to follow the bias-variance decomposition in Equation (2.11).

$$
\begin{aligned}
\text{Bias}(\hat{\rho}_{M,N}) &= \mathbb{E}\left[\hat{\rho}_{M,N}\right] - \rho \\
&= \mathbb{E}\left[\frac{1}{M}\sum_{i=1}^{M} h\left(\hat{L}_{N,i}\right)\right] - \mathbb{E}\left[h\left(L\right)\right] \\
&= \mathbb{E}\left[h\left(\hat{L}_N\right) - h\left(L\right)\right] \\
&= \mathbb{E}\left[h'\left(L\right)\left(\hat{L}_N - L\right) + \frac{1}{2}h''\left(z\right)\left(\hat{L}_N - L\right)^2\right], \qquad (2.16)
\end{aligned}
$$

where z_i is between L_i and $\hat{L}_{N,i}$, and the last equality follows from the Taylor expansion of the smooth function h. The first and second order Taylor expansion terms are analyzed separately in Equation (2.17) and Equation (2.18).

$$
\begin{aligned}
\mathbb{E}\left[h'\left(L\right)\left(\hat{L}_N - L\right)\right] &= \mathbb{E}\left[h'\left(L\right)\left(\hat{L}_N - L\right)\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[h'\left(L\right)\left(\hat{L}_N - L\right)|X\right]\right] \\
&= \mathbb{E}\left[h'\left(L\right)\mathbb{E}\left[\hat{L}_N - L|X\right]\right] = 0. \qquad (2.17)
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{E}\left[h''\left(z\right)\left(\hat{L}_N - L\right)^2\right] &\leq \mathbb{E}\left[\left|h''\left(z\right)\right|\left(\hat{L}_N - L\right)^2\right] \\
&\leq C_{h,2}\mathbb{E}\left[\left(\hat{L}_N - L\right)^2\right] \\
&= C_{h,2}\text{Var}(\hat{L}_N) = \mathcal{O}\left(N^{-1}\right). \qquad (2.18)
\end{aligned}
$$

It remains to analyze the variance term in Equation (2.11).

$$
\text{Var}(\hat{\rho}_{M,N}) = \frac{1}{M}\text{Var}(h(\hat{L}_N)) = \mathcal{O}(M^{-1}). \qquad (2.19)
$$

Equation (2.19) directly follows from Assumption 1 and the fact that $\hat{L}_N$ is a standard MC estimator of L. Combining Equation (2.17), 2.18, and 2.19, we are able to show the convergence order of the standard nested simulation procedure for a smooth function h without the cross term as in Equation (2.13).

Theorem 1 *Let h be a smooth function. MSE of the standard nested simulation procedure converges in order $\Gamma^{2/3}$, that is,*

$$\mathbb{E}\left[(\hat{\rho}_{M,N} - \rho)^2\right] = \mathcal{O}(\Gamma^{-2/3}).$$

The proof techniques used in deriving Theorem 1 is completely different from the one used in Gordy and Juneja (2010). The analysis in Gordy and Juneja (2010) is based on the differentiablity of the joint density of Y and the average inner simulation noise, which is difficult to verify in practice. Instead, we use a Taylor expansion of the smooth function h to analyze the convergence of the standard nested simulation procedure. The critical assumption in our derivation is Assumption 4, which is necessary to ensure the existence of the fourth moment of the simulation noise and the convergence of S_2, the second order term in the Taylor expansion. Assumption 4 is a moment condition that is easier to verify in practice than conditions on the joint density. As stated in Theorem **??**, the convergence in MSE automatically implies the convergence in probabilistic order.

Corollary 1 *Let h be a smooth function. The AE of the standard nested simulation procedure converges in probabilistic order $\Gamma^{-1/3}$, that is,*

$$|\hat{\rho}_{M,N} - \rho| = \mathcal{O}_{\mathbb{P}}(\Gamma^{-1/3}).$$

A Lipschitz Continuous Function h

We proceed to analyze the convergence for a Lipschitz continuous function h. For the CVaR analysis in Gordy and Juneja (2010), the authors assume the knowledge of the corresponding VaR. Hence, the analysis of CVaR is not complete. Instead, the quantity that is being analyzed is the mean excess loss, which corresponds to h being a hockey-stick function and belongs to the family of the Lipschitz continuous functions. Here we present the convergence analysis for the whole family of the Lipschitz continuous functions, which includes the hockey-stick function as a special case.

Assumption 5 *The function h is Lipschitz continuous. Hence, $|h(x_1) - h(x_2)| \leq K|x_1 - x_2|$ for some constant $K < \infty$.*

Assumption 5 is a standard assumption for analysis involving Lipschitz continuous functions, and it is also used in Broadie et al. (2015). For the Lipschitz continuous case, the first term in Equation (2.10) is analyzed differently.

$$\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h\left(\hat{L}_{N,i}\right) - h\left(L_i\right)\right)^2\right] \leq \mathbb{E}\left[\left(h\left(\hat{L}_N\right) - h\left(L\right)\right)^2\right]$$
$$\leq K^2\mathbb{E}\left[\left(\hat{L}_N - L\right)^2\right]$$
$$= K^2\mathbb{E}\left[\left(\bar{Z}_N(X)\right)^2\right] = \mathcal{O}(N^{-1}), \tag{2.20}$$

where the first inequality follows from Cauchy-Schwarz inequality, the second equality follows from Assumption 5, and the last equality follows from Assumption 2.

$$\mathbb{E}\left[(\hat{\rho}_{M,N} - \rho)^2\right] = \mathcal{O}(M^{-1}) + \mathcal{O}(N^{-1}) \tag{2.21}$$

Setting $M = \mathcal{O}(\Gamma^{1/2})$ and $N = \mathcal{O}(\Gamma^{1/2})$, we provide a looser bound than the one obtained for the hockey-stick h. Using the bias-variance decomposition in Equation (2.11), we obtain the following bias term.

$$\begin{aligned}
\text{Bias}(\hat{\rho}_{M,N}) &= \mathbb{E}\left[\hat{\rho}_{M,N}\right] - \rho \\
&= \mathbb{E}\left[\frac{1}{M}\sum_{i=1}^{M} h\left(\hat{L}_{N,i}\right)\right] - \mathbb{E}\left[h\left(L\right)\right] \\
&= \mathbb{E}\left[h\left(\hat{L}_N\right) - h\left(L\right)\right] \\
&\leq \mathbb{E}\left[\left|h\left(\hat{L}_N\right) - h\left(L\right)\right|\right] \\
&\leq K\mathbb{E}\left[\left|\hat{L}_N - L\right|\right] \\
&= K\mathbb{E}\left[\left|\bar{Z}_N\right|\right] \\
&\leq K\sqrt{\mathbb{E}\left[\left(\bar{Z}_N\right)^2\right]} = \mathcal{O}(N^{-1/2}).
\end{aligned} \tag{2.22}$$

The variance term in Equation (2.11) for Lipschitz continuous h is identical to the one in Equation (2.19).

$$\mathbb{E}\left[(\hat{\rho}_{M,N} - \rho)^2\right] = \mathcal{O}(M^{-1}) + \mathcal{O}(N^{-1}). \tag{2.23}$$

Setting $M = \mathcal{O}(\Gamma^{2/3})$ and $N = \mathcal{O}(\Gamma^{1/3})$, we provide the same rate of convergence as obtained for other risk measures in Gordy and Juneja (2010).

Theorem 2 *Let h be a Lipschitz continuous function. MSE of the standard nested simulation procedure converges in order $\Gamma^{2/3}$, that is,*

$$\mathbb{E}\left[(\hat{\rho}_{M,N} - \rho)^2\right] = \mathcal{O}(\Gamma^{-1/2}).$$

Immediately from Theorem **??**, the convergence in MSE automatically implies the convergence in probabilistic order.

Corollary 2 *Let h be a Lipschitz continuous function. AE of the standard nested simulation procedure converges in probabilistic order $\Gamma^{-1/3}$, that is,*

$$|\hat{\rho}_{M,N} - \rho| = \mathcal{O}_{\mathbb{P}}(\Gamma^{-1/4}).$$

2.3.3 Asymptotic Analysis of a kNN-based Nested Simulation Procedure for a Smooth Function

In Hong et al. (2017), the authors analyze the asymptotic convergence in terms of MSE for a kernel smoothing-based nested simulation procedure. The analysis is complete for the nested expectation form where h is either a smooth function, an indicator function, or a hockey-stick function. The kernel function of interest in the asymptotic analysis is the Nadaraya-Watson kernel. Nevertheless, the numerical results are derived based on a kNN-based nested simulation procedure. In this section, we attempt to fill in the holes in the analysis of Hong et al. (2017) by providing asymptotic convergence of a kNN-based nested simulation procedure for smooth h in the nested expectation form. Surprisingly, the asymptotic convergence rate of the kNN-based nested simulation procedure is noticeably different from the result obtained in Hong et al. (2017).

A kNN regression can be used as a non-parametric metamodel that estimates the conditional expectation of Y given X by averaging the Y values of the k nearest neighbors of X. Without loss of generality, we assume that $\{(X_i, \bar{Y}_{n,i}), i = 1, \ldots, M\}$ are i.i.d. observations of (X, Y) from the standard nested simulation procedure with an inner simulation budget of n, where X_i is the i^{th} of the outer scenario $X \in \Omega \subset \mathbb{R}^d$, and $\bar{Y}_{n,i}$ is an estimate of $Y|X_i$ with an inner simulation budget of n. Let $f_{XY}(x, y)$ be a joint density of random vector

(X, Y), and $f_X(x) = \int f_{XY}(x, y)dy$ be a marginal density of X. Let k be a sequence of positive integers $k = k(M)$ such that

$$\begin{aligned} k &\to \infty, \\ \frac{k}{M} &\to 0, \quad \text{as } M \to \infty, \\ \frac{\log(M)}{k} &\to 0, \quad \text{as } k \to \infty. \end{aligned}$$

A kNN-based nested simulation procedure with a model parameter k estimates the risk measure $\rho = \mathbb{E}[h(L)]$ with

$$\hat{L}_{M,N}^{\text{kNN}}(x) = \frac{\frac{1}{MR_M^d} \sum_{i=1}^{M} \hat{L}_{N,i} K\left(\frac{x - X_i}{R_M}\right)}{\frac{1}{MR_M^d} \sum_{i=1}^{M} K\left(\frac{x - X_i}{R_M}\right)}$$

where R_k is a Euclidean distance between X_i and its k-th nearest neighbor, and $K : \mathbb{R}^d \mapsto \mathbb{R}$ is a bounded, nonnegative kernel function satisfying

$$\begin{aligned} \int K(u)du &= 1, \\ K(u) &= 0 \quad \text{for } \|u\| \geq 1, \end{aligned}$$

where $\|\cdot\|$ is a Euclidean norm, and k is a number of nearest neighbors to consider. For a kNN estimator trained with M i.i.d. observations, denote $B_M(x)$ and $V_M(x)$ as its bias and variance at point x, respectively.

Assumption 6 *f_X is bounded and continuously differentiable up to a second order in a neighborhood of x with $f_X(x) > 0$, f_X, $\mathbb{E}[Y^2|X = x] < \infty$, and the kernel function K satisfies*

$$\begin{aligned} \int \|u\|^2 |K(u)|du &< \infty, \\ \int \nu_\alpha K(u)du &= 0, \quad \textit{for } \alpha = 1, \ldots, d. \end{aligned}$$

Suppose that $\mathbb{P}(\|x - X\| > r) = \mathcal{O}(r^{-\alpha})$ for some $\alpha > 0$ as $r \to \infty$.

The following lemma from Mack (1981) characterizes pointwise convergence of the kNN estimator in terms of its bias and variance at point x.

Lemma 1 *Assume that a random vector (X, Y) and a kernel function K satisfies Assumption 6. Then the bias and variance of the kNN estimator at point x are given by*

$$B_M(x) = A_{kNN}(x) \cdot \left(\left(\frac{k}{M} \right)^{\frac{2}{d}} + o\left(\frac{k}{M} \right)^{\frac{2}{d}} \right) + B_{kNN} \cdot \left(\frac{1}{k} + o\left(\frac{1}{k} \right) \right),$$
$$V_M(x) = \frac{C_{kNN} \cdot Var[Y|X = x]}{k} \int K^2(u)du + o\left(\frac{1}{k} \right).$$

Let h be a smooth function, i.e., Assumption 3 holds. The bias of the estimator of ρ for a kNN-based nested simulation procedure is analyzed following a Taylor expansion.

$$\begin{aligned} &\mathbb{E}\left[h(\hat{L}_{M,n}^{\text{kNN}}(X)) - h(L) \right] \\ &= \mathbb{E}\left[h'(L)\left(\hat{L}_{M,n}^{\text{kNN}}(X) - L \right) + \frac{1}{2} h''(z)\left(\hat{L}_{M,n}^{\text{kNN}}(X) - L \right)^2 \right], \end{aligned} \tag{2.24}$$

where the bias is decomposed into two terms, and z is between $\hat{L}_{M,n}^{\text{kNN}}(X)$ and L. The first term in Equation (2.24) is

$$\begin{aligned} &\mathbb{E}\left[h'(L)\left(\hat{L}_{M,n}^{\text{kNN}}(X) - L \right) \right] \\ &= \mathbb{E}\left[h'(L)\mathbb{E}\left[\hat{L}_{M,n}^{\text{kNN}}(X) - L|X \right] \right] \\ &= \int_\Omega h'(L) B_M(x) f_X(x) dx \\ &= \int_\Omega h'(L) \left(A_{\text{kNN}}(x) \left(\frac{k}{M} \right)^{2/d} (1 + o_x(1)) + B_{\text{kNN}} \cdot \left(\frac{1}{k} + o_x\left(\frac{1}{k} \right) \right) \right) f_X(x) dx \\ &= \left(\frac{k}{M} \right)^{2/d} \mathbb{E}\left[h'(L) A_{\text{kNN}}(X) \right] (1 + o(1)) + \frac{B_{\text{kNN}}}{k} \mathbb{E}\left[h'(L) \right] (1 + o(1)). \end{aligned} \tag{2.25}$$

The second term in Equation (2.24) is

$$
\begin{aligned}
&\mathbb{E}\left[h''(z)\left(\hat{L}_{M,n}^{\mathrm{kNN}}(X)-L\right)^2\right] \\
&=\int_\Omega h''(z)\left(B_M^2(X)+V_M(x)\right)f_X(x)dx \\
&\leq C_2\int_\Omega\left(\frac{C_{\mathrm{kNN}}\cdot\mathrm{Var}[Y|X=x]}{k}\int K^2(u)du+o\left(\frac{1}{k}\right)\right)f_X(x)dx \\
&+C_2\int_\Omega\left(A_{\mathrm{kNN}}(x)\left(\frac{k}{M}\right)^{2/d}(1+o_x(1))+B_{\mathrm{kNN}}\cdot\left(\frac{1}{k}+o_x\left(\frac{1}{k}\right)\right)\right)^2 f_X(x)dx, \qquad (2.26)
\end{aligned}
$$

where the second term in Equation (2.26) is of higher order than Equation (2.25), and it is negligible since $kM^{-1}\to 0$ as $M\to\infty$ and $k\to\infty$, The first term in Equation (2.26) is $\mathcal{O}\left(\frac{1}{k}\right)$. Set $k=\mathcal{O}(M^{\frac{2}{2+d}})$ and $M=\mathcal{O}(\Gamma)$, the bias of the kNN-based nested simulation procedure converges in the order of $\Gamma^{-\frac{2}{2+d}}$.

Definition 6 *Let h be a smooth function. The bias of the kNN-based nested simulation procedure converges in the order of $\Gamma^{-\frac{2}{2+d}}$, that is,*

$$\mathbb{E}\left[h(\hat{L}_{M,n}^{kNN}(X))-h(L)\right]=\mathcal{O}(\Gamma^{-\frac{2}{2+d}}).$$

Definition 6 is obtained by matching the orders of the bias terms in Equation (2.25) and 2.26. It is the tightest bound that can be obtained for the convergence of the kNN-based nested simulation procedure with a smooth function h. The variance is left for future work. It implies that the kNN-based nested simulation procedure converges at most in the order of $\Gamma^{-\frac{2}{3}}$ for $d=1$ and has slower convergence than the standard procedure for $d\geq 2$. Asymptotically, it has slower convergence than the result obtained in Hong et al. (2017) for the Nadaraya-Watson kernel. This discrepancy underscores the critical importance of aligning theoretical analysis with numerical illustrations. The slower convergence rate observed with the kNN approach not only challenges the robustness of empirical findings derived under differing assumptions but also signals a cautionary note for practitioners. This alignment is crucial for ensuring the reliability and validity of conclusions drawn from such procedures, especially in the context of risk management and financial decision-making.

2.4 Convergence Orders and Critical Assumptions of Nested Simulation Procedures

In this section, we summarize the convergence orders and the critical assumptions of the nested simulation procedures. The results are summarized in 2.3, where the orders of convergence are presented in the order of the total simulation budget Γ.

Estimator for L	**Smooth h**	**Lipschitz / hockey-stick h**	**Indicator h**	**VaR**	**CVaR**
Standard MC	$\mathcal{O}(\Gamma^{-2/3})$	$\mathcal{O}(\Gamma^{-1/2})$ / $\mathcal{O}(\Gamma^{-2/3})$	$\mathcal{O}(\Gamma^{-2/3})$	$\mathcal{O}(\Gamma^{-2/3})$	$\times$
Multi-level MC	$\times$	$\times$ / $\times$	$\mathcal{O}(\Gamma^{-1}\lvert\log(\Gamma)\rvert)$	$\times$	$\times$
Regression	$\mathcal{O}(\Gamma^{-1})$	$\mathcal{O}(\Gamma^{-1})/\mathcal{O}(\Gamma^{-1})$	$\times$	$\times$	$\times$
Kernel smoothing	$\mathcal{O}(\Gamma^{-\min\{1,\frac{4}{d+2}\}})$	$\times$ / $\mathcal{O}(\Gamma^{-\min\{1,\frac{4}{d+2}\}})$	$\mathcal{O}(\Gamma^{-\min\{1,\frac{4}{d+2}\}})$	$\times$	$\times$
Kernel ridge regression	$\mathcal{O}(\Gamma^{-1})$	$\times$ / $\mathcal{O}(\Gamma^{-1})$	$\mathcal{O}(\Gamma^{-1})$	$\mathcal{O}(\Gamma^{-1})$	$\mathcal{O}(\Gamma^{-1})$
Likelihood ratio	$\mathcal{O}(\Gamma^{-1})$	$\times$ / $\mathcal{O}(\Gamma^{-1})$	$\mathcal{O}(\Gamma^{-1})$	$\times$	$\times$

Table 2.3: Asymptotic rate of convergence of nested simulation procedures in MSE

The asymptotic convergence rate of the nested simulation procedures is highly dependent on the assumptions on the random variable X, Y, and the inner simulation noise. As more advanced supervised learning techniques are used to estimate the function g, a more restrictive set of assumptions is required to ensure the asymptotic convergence of the inner estimators. In this section, we summarize the pivotal assumptions regarding the random variables and the inner simulation noise for all nested simulation procedures. They are presented in an order reflecting their relative ease in satisfying the assumptions.

2.4.1 Standard Assumptions

Gordy and Juneja (2010) and Broadie et al. (2015) assume that the inner simulation noise has zero mean and variance that is inversely proportional to the inner simulation budget N, i.e., Assumption 2. The analysis in Gordy and Juneja (2010) further requires assumptions on $f_{Y,\tilde{Z}_N}(y, z_N)$, a joint density of random variable Y and the normalized inner simulation noise $\tilde{Z}_N := \sqrt{N} \cdot \bar{Z}_N(X)$.

2.4.2 Assumptions on Joint Density

Assumption 7 *The joint density $f_{Y,\tilde{Z}_N}(y, z_N)$ and its partial derivatives $\frac{\partial f_{Y,\tilde{Z}_N}(y,z_N)}{\partial y}$ and $\frac{\partial^2 f_{Y,\tilde{Z}_N}(y,z)}{\partial y^2}$ exist.*

Assumption 8 *For $N \geq 1$, there exist non-negative functions $p_{0,N}(\cdot)$, $p_{1,N}(\cdot)$, and $p_{2,N}(\cdot)$ such that for all y and z,*

$$\begin{aligned} f_{Y,\tilde{Z}_N}(y,z) &\leq p_{0,N}(z) \\ \left|\frac{\partial f_{Y,\tilde{Z}_N}(y,z)}{\partial y}\right| &\leq p_{1,N}(z) \\ \left|\frac{\partial^2 f_{Y,\tilde{Z}_N}(y,z)}{\partial y^2}\right| &\leq p_{2,N}(z). \end{aligned}$$

In addition,

$$\sup_N \int_{-\infty}^{\infty} |z|^r p_{i,N}(z) dz < \infty \tag{2.27}$$

for $i = 0, 1, 2$ and $r \in [0, 4]$.

Assumption 7 and 8 are necessary for the Taylor expansion up to the second order on the joint density $f_{Y,\tilde{Z}_N}(y, z_N)$, and they can easily be satisfied by perturbing Y and $\tilde{Z}_N$ with a zero-mean normal random variable with a small variance. However, Assumption 8 is hard to be verified in practice. Since Broadie et al. (2015) only analyze the nested expectation case with a smooth and a Lipschitz continuous function h, the regression-based nested simulation procedure does not require Assumptions 7 and 8 for the convergence analysis. In our analysis of the standard nested simulation procedure for a smooth function h in the nested expectation form, we make additional assumptions on the fourth moment of the inner simulation noise, i.e., Assumption 3, so that the second-order term in the Taylor expansion can be bounded.

2.4.3 Assumptions for the MLMC Procedure

In addition to Assumption 8, the MLMC procedure requires assumptions on the random variable $Q := \frac{|\mathbb{E}[Y|X]|}{\mathrm{Var}[Y|X]}$.

Assumption 9 *f_Q, the density of Q exists, and there exist positive constants q_0 such that $f_Q(q) \leq q_0$ for all $q \in [0, q_0]$.*

Assumption 9 ensures that the conditional expectation of Y given X is not concentrated around zero, as Giles and Haji-Ali (2019) show the asymptotic convergence of the MLMC procedure only when ρ is in the nested expectation form with h being an indicator function, i.e., $h(x) = \mathbb{I}_{\{x \geq 0\}}$.

2.4.4 Assumptions for the Likelihood Ratio-Based Procedure

In addition to Assumption 8, the likelihood ratio-based nested simulation procedure requires assumptions on the marginal density of H, the random variable that characterizes the stochasticity of the inner simulation model. More specifically, for the likelihood ratio-based nested simulation procedure, the random variable H is introduced to represent the inner simulation noise, and Y can be directly expressed as a function of H and X, i.e., $Y = Y(H, X)$. Instead of simulating Y directly, the random variable H is simulated from the conditional distribution $f_{H|X}(y|x)$, and Y is then obtained by evaluating $Y(H, X)$.

Assumption 10 *The marginal density $f_H(y)$ of H exists. $f_H(y)$ can be sampled and evaluated, and $Y(y, x)f_{H|X}(y|x) = 0$ whenever $f_H(x) = 0$.*

Furthermore, the likelihood ratio-based nested simulation procedure requires the samples of H to be independent with the outer scenario X.

Assumption 11 *The inner sample $H \sim f_H(y)$ is independent of X. Samples $\{X_i, i = 1, \ldots, M\}$ and $\{H_i, i = 1, \ldots, N\}$ are i.i.d. samples from $f_X(x)$ and $f_H(y)$, respectively.*

Assumption 11 is necessary for $\hat{L}_N^{\mathrm{LR}}(X_i)$ to be an unbiased and strongly consistent estimator of L_i, which is a necessary condition for the convergence of $\hat{\rho}_{M,N}^{\mathrm{LR}}$, the likelihood ratio-based nested simulation estimator of ρ. Assumption 10 and Assumption 11 are likely to be satisfied in practice, as one can simulate H separately from X and evaluate $Y(H, X)$ for each pair of (H, X).

2.4.5 Assumptions for the Kernel-Based Procedure

For the kernel-based nested simulation procedure, Hong et al. (2017) simplifies the analysis by assuming that for each X_i, the inner simulation budget N is fixed together with the variance of the inner simulation noise $\bar{Z}_{N,i}$. In essence, this fixed variance design treats the $(X_i, \bar{Y}_{N,i})$ as i.i.d. observations of (X, Y), and the kernel-based nested simulation procedure is analyzed as a non-parametric regression problem. Regularity conditions on $f_X(x)$, the density of X, and the conditional second moment of Y are assumed to ensure the convergence of $\hat{L}_{M,N}^{\mathrm{KS}}(X)$, the kernel smoothing estimator of L.

Assumption 12 *$f_X(x)$ and $\mathbb{E}[Y^2|X = x]$ exist, $f(x) > 0$, and $f(x)$ is thrice continuously differentiable at x with bounded third-order derivative.*

Further assumptions are made on the kernel function K_w and the bandwidth w to ensure the convergence of the kernel smoothing estimator $\hat{L}^{\mathrm{KS}}_{M,N}(X)$. A similar set of assumptions is made in Jennen-Steinmetz and Gasser (1988) for the convergence analysis of a kernel smoothing estimator with kernel functions of higher orders.

2.4.6 Assumptions for the KRR-Based Procedure

For the VaR-based nested simulation procedure, the convergence is guaranteed only when $\bar{Z}_N(X)$ is the sum of N i.i.d. zero-mean sub-Gaussian random variables. The sub-Gaussian assumption imposes a stronger condition on the inner simulation noise than all the other nested simulation procedures. The inner simulation noise is assumed to have lighter tails than the normal distribution, while examples of random variables that satisfy the sub-Gaussian assumption are hard to find. In addition, the following assumptions are made on Ω, which is the domain of the outer scenario X:

Assumption 13 *Ω is a bounded subset of $\mathbb{R}^d$, and $f_X(x)$ is bounded above and below away from zero.*

Assumption 13 rules out the possibility of the outer scenario X being a normal random variable, which is a common assumption in the literature. Furthermore, the convergence analysis of the VaR-based nested simulation procedure is shown in terms of AE rather than in MSE, which is due to the complexity of the KRR metatmodel.

2.5 Finite-Sample Experiments

The theoretical framework has lent itself for a comparison of the asymptotic convergence behavior for different nested simulation procedures. However, a simulation budget in practice is almost always finite. In this section, we conduct a series of numerical experiments to compare the empirical convergence of the nested simulation procedures for different risk measures, option types, and asset dimensions. Numerical experiments are conducted on portfolios that consist of options on d underlying assets, whose dynamics follow a multidimensional geometric Brownian motion with 0.3 pairwise correlation. The initial values of the assets are set to be 100. The drifts are set to be 0.08, and the risk free rate is set to be 0.05.

A total of 5 nested simulation procedures are considered, namely a standard nested simulation procedure, a regression-based, a kernel smoothing-based, a likelihood ratio-based, and a KRR-based nested simulation procedure. For the standard nested simulation procedure, the bootstrap-based budget allocation strategy from Zhang et al. (2021) is implemented to estimate the optimal values of outer and inner simulation budgets. For the regression metamodel, the Laguerre polynomials up to degree 3 are used as the default basis functions. The kNN metamodel is implemented for the kernel smoothing-based procedure, and a cross-validation procedure with a grid search is used to find the optimal number of neighbors. The VaR metamodel is implemented with a Matérn kernel, and the smoothness, the length scale, and the regularization hyperparameters are found by a cross-validation procedure with a Bayesian search (Frazier, 2018). Due to budget constraints, the hyperparameters of the non-parametric regression metamodel are tuned and used across all macro replications of the same experiment.

The procedures are compared for 5 types of risk measures:

- a quadratic tracking error,
- a mean excess loss over a threshold u,
- a probability of a large loss over a threshold u,
- the VaR, and
- the CVaR,

where the threshold u for the mean excess loss and the probability of a large loss is set to be 90% VaR. The procedures are compared for different portfolios consisting of different types of options:

- Portfolio 1 consists of d assets. This portfolio contains 3 European call options written on each asset with strikes 90, 100, and 110, respectively.
- Portfolio 2 consists of d assets. This portfolio contains 3 geometric Asian options written on each asset with strikes 90, 100, and 110, respectively.
- Portfolio 3 consists of d assets. This portfolio contains 3 up-and-out barrier call options written on each asset with strikes 90, 100, and 110, respectively. They have a barrier level of 120.

- Portfolio 4 considers d assets. This portfolio contains 3 down-and-out barrier call options written on each asset with strikes 90, 100, and 110, respectively. They have a barrier level of 90.

- Portfolio 5 contains 1 asset. This portfolio longs 2 down-and-out barrier put options and shorts 1 barrier down-and-out put option.

Except for Portfolio 5, 5 different asset dimensions are considered, i.e., $d = 1, 2, 5, 10, 20$. All options have a maturity of 1 year, and we are interested in the value of the portfolio at time 3/50 year from now. 525 experiment settings that arise from an exhaustive combination of the above simulation procedures, risk measures, portfolios, and asset dimensions are generated. For each experiment setting, the estimation of the risk measure is repeated 1000 times to obtain the empirical MSE of the corresponding estimator under a range of simulation budgets. The total simulation budget Γ ranges from 10,000 to 10,240,000 in a geometric progression with a common ratio of 2. The empirical convergence results of each estimator are measured and recorded to illustrate their empirical behavior under different risk measures, option types, and asset dimensions. All numerical experiments are conducted on a linux server with four Intel Xeon Gold 6230 20-core 2.1 GHz CPU with 768 GB RAM.

We start by examining the empirical convergence results for the most basic case, i.e., a quadratic tracking error risk measure for European call options on Portfolio 1 with $d = 1$.

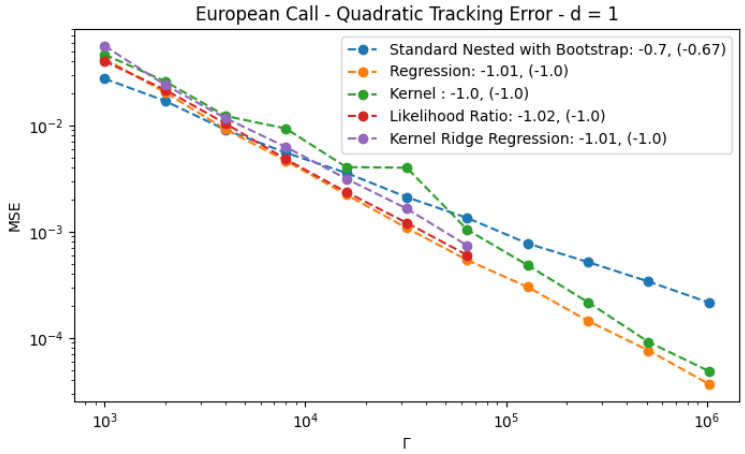


Figure 2.1: Empirical convergence of nested simulation procedures for quadratic tracking error on Portfolio 1 with $d = 1$

In Figure 2.1, the MSEs of the nested simulation procedures are plotted against the total simulation budget Γ in a log-log scale, where each point represents the average MSE of the corresponding estimator over 1000 macro replications of the same experiment. Due to the difficulty of fixing Γ for the MLMC procedure, the empirical rate of convergence of the MLMC procedure is not reported here, but a detailed analysis of the empirical convergence of the MLMC procedure is provided in Section 2.5.7.

A regression line is fitted to the MSEs of each nested simulation procedure against the simulation budget Γ in a log-log scale, and the slope of the fitted line is reported. The slopes of the fitted lines can be interpreted as the empirical rates of convergence. Alongside in the parentheses are the asymptotic rates of convergence obtained by the theoretical analysis in Section 2.3. By comparing the empirical rates of convergence with the asymptotic rates, we can observe how well the empirical rates of convergence match their asymptotic rates. In the case illustrated in Figure 2.1, the empirical rates of all procedures closely match their asymptotic rates.

Except for the standard nested simulation procedure, the empirical rates of convergence of the other procedures are close to Γ^{-1}, i.e., all of them achieve similar finite-sample performance that is better than the standard procedure. Pooling of inner simulation samples across different outer scenarios is the key reason behind their higher empirical rates of convergence. However, pooling also introduces additional computational cost that is not reflected in the simulation budget Γ. Due to the computational complexity of the likelihood ratio-based and the VaR-based procedures, their MSEs are not reported for Γ larger than 16,000 and 64,000, respectively. Detailed discussion of the additional computational cost of pooling is provided in Section 2.6.

In the following sections, we examine the empirical convergence of the nested simulation procedures similarly as in Figure 2.1. With a more detailed analysis of different risk measures, option types, and asset dimensions, we aim at providing a comprehensive understanding and comparison of the convergence behavior of the nested simulation procedures in practice.

2.5.1 Sensitivity to the Asset Dimension

In portfolio risk management, the asset dimension is a critical factor that determines the complexity of the portfolio and the computational cost of the risk measure estimation. The theoretical analyses in Section 2.4 suggest that the asymptotic rate of convergence of the nested simulation procedures, except for the kernel smoothing-based procedure, is independent of the asset dimension. However, in practice, their empirical convergence behavior

could be different. In this section, we examine the sensitivity to the asset dimension of the empirical convergence of the nested simulation procedures.

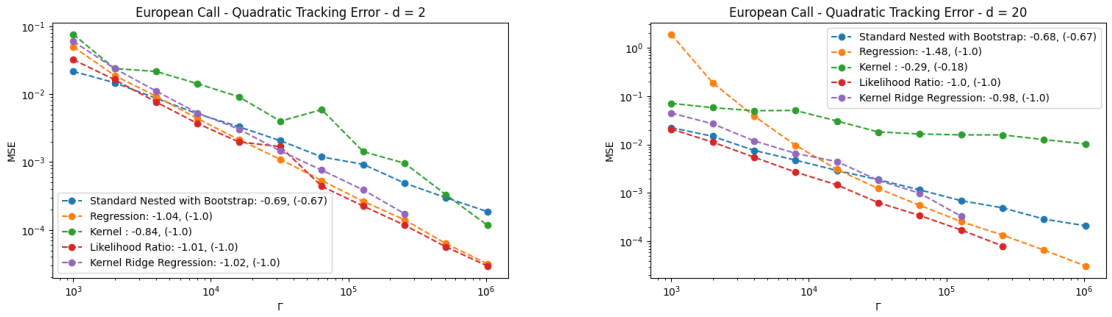


Figure 2.2: Empirical convergence of nested simulation procedures for quadratic tracking error on Portfolio 1 with different asset dimensions

In Figure 2.2, the empirical convergence of the nested simulation procedures for a quadratic tracking error on Portfolio 1 is illustrated for different asset dimensions, i.e., $d = 2$ and $d = 20$. The empirical rates of convergence of the standard, the VaR-based, and the likelihood ratio-based procedures closely match their asymptotic rates for both asset dimensions, and they are insensitive to an increase in the asset dimension. In contrast, both the kernel smoothing-based and regression-based procedures are greatly affected by the asset dimension. Their sensitivities to the asset dimension differ in finite-sample experiments:

- For the kernel smoothing-based procedure, the empirical rate of convergence decreases as the asset dimension increases. This observation aligns with the theoretical analysis in Section 2.3, which demonstrates that the asymptotic rate of convergence of the kernel smoothing-based procedure is sensitive to the asset dimension. Nevertheless, even for $d = 2$ and $d = 20$, the empirical rate remains higher than its asymptotic rate.

- For the regression-based procedure, the asymptotic rate is not affected by the asset dimension, but the empirical rates of convergence are higher than the asymptotic rates for both asset dimensions, and the empirical rate is higher for $d = 20$ than for $d = 2$.

In the following sections, we provide detailed explanations for the discrepancy between their empirical rates of convergence and their asymptotic rates.

2.5.2 Empirical Convergence of Kernel Smoothing-Based Procedures

Kernel smoothing is a non-parametric regression metamodel. According to the theoretical analysis in Section 2.3, the asymptotic rate of convergence of the kernel smoothing-based nested simulation procedure is highly dependent on the asset dimension d. In this section, we use a kNN metamodel for the kernel smoothing-based nested simulation procedure, and we examine its empirical convergence for different asset dimensions.

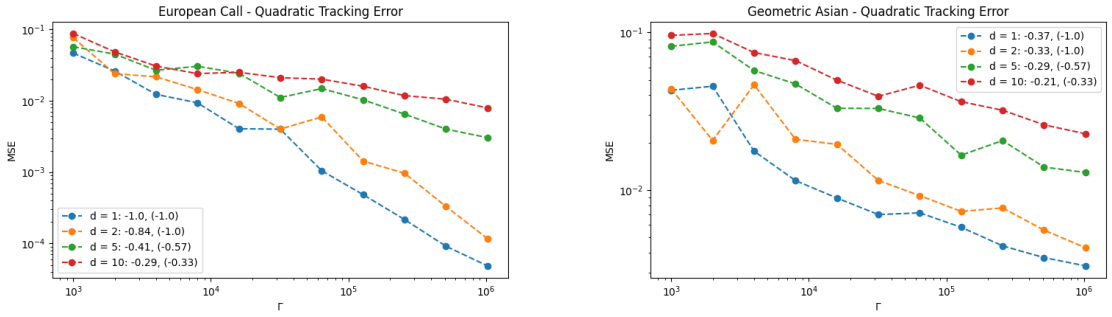


Figure 2.3: Empirical convergence of kernel smoothing procedure for different values of d

In Figure 2.3, the empirical convergence of the kernel smoothing-based nested simulation procedure for a quadratic tracking error is illustrated for different asset dimensions, i.e., $d = 1, 2, 5, 10$. The experiment finds that the kernel smoothing-based nested simulation procedure is extremely sensitive to the asset dimension and the payoff structure. While the asset dimension is expected to be a critical factor as shown in the theoretical analysis, the payoff structure is an unexpected factor that affects the empirical convergence of the kernel smoothing-based nested simulation procedure. For a portfolio with geometric Asian options, the payoff complexity is higher than that of a portfolio with European call options. The empirical rate of convergence of the kernel smoothing-based procedure is substantially lower for the portfolio with geometric Asian options than for the portfolio with European call options. For geometric Asian options, the empirical rates of convergence are even lower than the asymptotic rates for all asset dimensions.

Due to the computational cost of the kernel smoothing-based nested simulation procedure, we are not able to conduct experiments for higher budget levels. However, we are able to conduct additional experiments to examine the effects of cross-validation on the empirical convergence of the kernel smoothing-based procedure. Another phenomenon that is observed in Figure 2.3 is that the empirical rate of convergence of the kernel smoothing-based procedure does not decrease monotonically as the simulation budget increases. This

is likely due to the fact that the kernel smoothing-based procedure, as a non-parametric regression procedure, is highly dependent on the cross-validation of the hyperparameters of the metamodel. For the kernel smoothing-based procedure, the kNN metamodel is implemented. Hence, the hyperparameter of interest is the number of nearest neighbors k. In our numerical experiment, cross-validation is conducted once to select the optimal value of k for each simulation budget, and the selected value of k is fixed for all 1000 replications. Therefore, a suboptimal selection of the optimal value of k can lead to a poor metamodel estimate, and the MSE of the kRR-based procedure will be dominated by the model error of the metamodel.

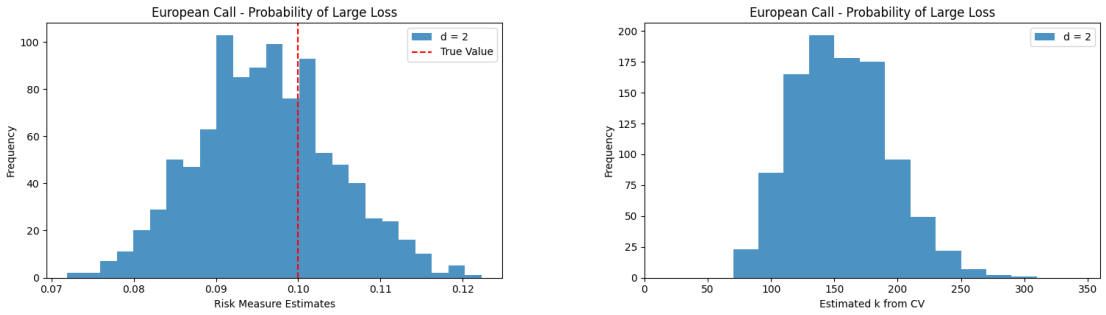


Figure 2.4: Cross-validation for the kernel smoothing-based procedure with $\Gamma = 100{,}000$

In Figure 2.4, the empirical convergence of the kernel smoothing-based nested simulation procedure for the probability of large loss is illustrated for different values of k with $\Gamma = 100{,}000$. The observation is that the value of optimal k estimated by cross-validation is highly variable across different replications. For $d = 2$, the optimal value of k is estimated to be between 120 to 220 for most replications, and the estimated risk measures are close to the true risk measure of 0.1. For the two replications where $k = 80$ is estimated as the optimal value of k, the resulting estimates of risk measures are 0.1187 and 0.1189, which are substantially higher than the estimated risk measures for the other replications and far from the true risk measure of 0.1. This observation suggests that the empirical convergence of the kernel smoothing-based nested simulation procedure is highly sensitive to the cross-validation of the hyperparameters of the metamodel. A VaR-based nested simulation procedure suffers from the same problem. The fluctuation in Figure 2.3 is likely due to poor cross-validation of the hyperparameters of the metamodel.

2.5.3 Empirical Convergence of Parametric Regression Procedures

For both the regression-based and kernel smoothing-based procedures, the observed higher empirical rates of convergence can be explained by the poor performance of their metamodels at low simulation budgets. When simulation budgets are low, the metamodels trained to approximate the true inner simulation are inadequate. The MSEs for these procedures are dominated by the model errors of the metamodels. In other words, they have not yet reached their asymptotic convergence regimes.

Due to computation constraints, we are not able to conduct experiments for kernel smoothing-based nested simulation procedures for higher budget levels, but we are able to conduct additional experiments for the regression-based nested simulation procedure. In our previous numerical experiments, the empirical rate of convergence of the regression procedure is observed to be much larger than its asymptotic rate of convergence. For dimensions larger than 10, the MSE of the regression procedure decreases quickly in the beginning, and its rate of decrease stabilizes after a certain budget level. In Figure 2.5 illustrates the empirical convergence of the regression-based procedure in more detail at higher simulation budgets, where the asymptotic level of convergence is reached for $d = 20$.

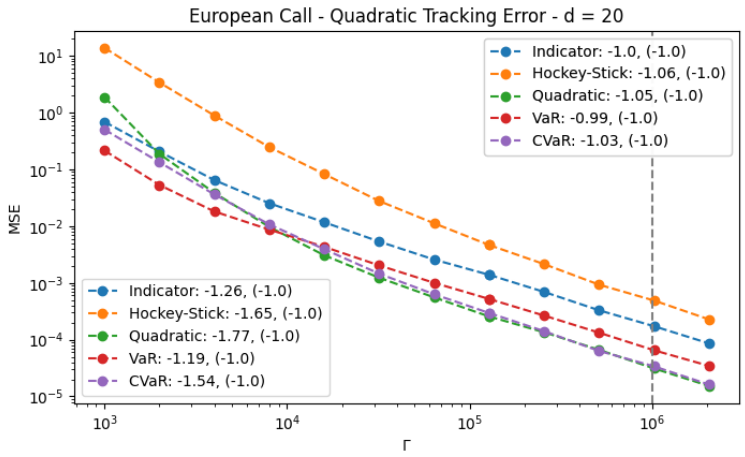


Figure 2.5: Empirical convergence of regression procedure for European call options and $d = 20$

The left part of Figure 2.5 contains the MSEs of the regression procedure for budget sizes that are smaller than 10^3. Slopes of the fitted lines on the left correspond to

the empirical rate of convergence of the regression procedure for budget levels between 10^3 and 10^4. To investigate the convergence behavior for higher budget levels, we conduct additional experiments for the European call option with a dimension of 20. The additional experiments are summarized on the right side of Figure 2.5. After reaching a certain budget level, i.e., $\Gamma = 10^6$ in our case, the empirical rate of convergence for the regression-based nested simulation procedure approaches its asymptotic rate. For nested simulation procedures with a biased metamodel, the metamodel estimates of the true inner simulation are poor, especially for smaller budget sizes. We are able to clearly observe this phenomenon for the regression metamodel, and it can be explained by dividing the MSE into metamodeling bias and simulation variance.

- For small budget sizes, the improvement in the bias of the regression metamodel dominates the improvement in the simulation variance. Performing poorly for extremely low simulation budgets, the regression metamodel improves dramatically as the simulation budget increases.
- As the simulation budget gets higher, the improvement of the regression bias becomes negligible compared with that of the simulation variance. The regression metamodel ceases to improve after reaching a certain level ($\Gamma = 10^6$ in our case), and the improvement of simulation variance dominates.

Due to high computational costs, we are not able to conduct experiments for higher budget levels, but we expect the MSE of the regression-based nested simulation procedure to stabilize after $\Gamma = 10^6$, and the empirical rate of convergence to approach its asymptotic rate. The empirical convergence rates reported in the graph legends are higher than the actual rates after regression-based nested simulation procedure reaches the asymptotic convergence regime. In the context of nested simulation procedures, a parametric regression metamodel has several advantages over a non-parametric regression metamodel.

- Parametric regression achieves a good empirical convergence behavior for a moderate simulation budget.
- Parametric regression is insensitive to changes in the asset dimension.
- Parametric regression is easier to implement. It does not require cross-validation to select the hyperparameters of the metamodel.

However, parametric regression suffers from the model misspecification risk. More specifically, selecting a wrong regression basis can lead to a poor metamodel estimate, where

the MSE of the nested simulation procedure is dominated by the model error of the metamodel (Broadie et al., 2015).

In our previous numerical experiments, the regression-based nested simulation procedure is implemented with Laguerre polynomials up to degree 3 as the basis functions. It is a standard choice in the literature. To examine the sensitivity of the empirical convergence behavior to the regression basis, we consider different basis functions, i.e., regular polynomial regression up to degree 3.

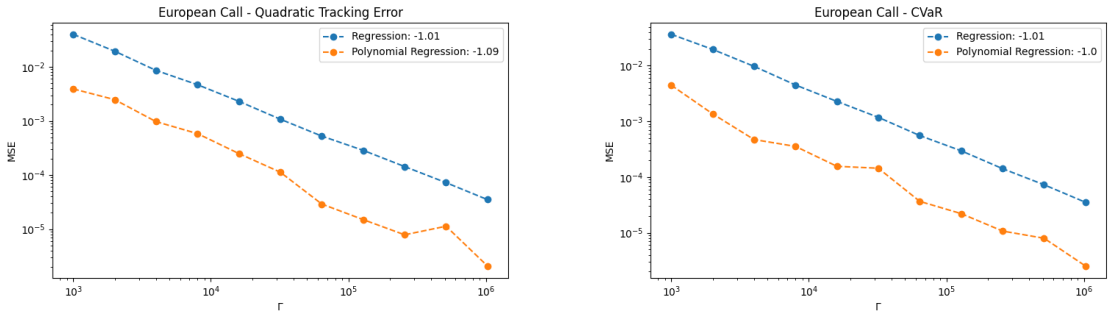


Figure 2.6: Empirical convergence of regression-based nested simulation procedures for different regression bases

In Figure 2.6, the empirical convergence results of the regression-based nested simulation procedure for a quadratic tracking error and a 90%-CVaR are illustrated for the two different regression bases. The empirical rates of convergence of the regression-based nested simulation procedure are observed to be insensitive to the chosen regression basis.

2.5.4 Sensitivity to the Option Types and Risk Measures

In our previous numerical experiments, we have examined the empirical convergence behavior of the nested simulation procedures for European call options, which only depends on the asset price at maturity. To examine the sensitivity of the empirical convergence behavior to the option type, we consider path-dependent options, which are geometric Asian options and barrier options.

In Figure 2.7, the empirical convergence of the nested simulation procedures for the quadratic tracking errors of Portfolio 1 and Portfolio 4 is illustrated for $d = 20$. The empirical rates of standard, likelihood ratio-based, and VaR-based nested simulation procedures closely ensemble their asymptotic rates for path-dependent options. The empirical rate of

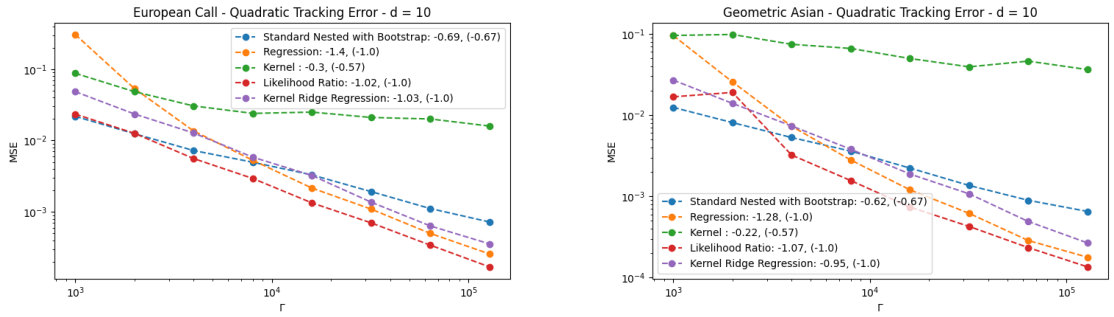


Figure 2.7: Empirical convergence of nested simulation procedures for quadratic tracking error on different portfolios with $d = 20$

convergence of the regression-based and kernel smoothing-based procedures is much higher than their asymptotic rates for barrier options. The reasons are explained in Section 2.5.3 and Section 2.5.2.

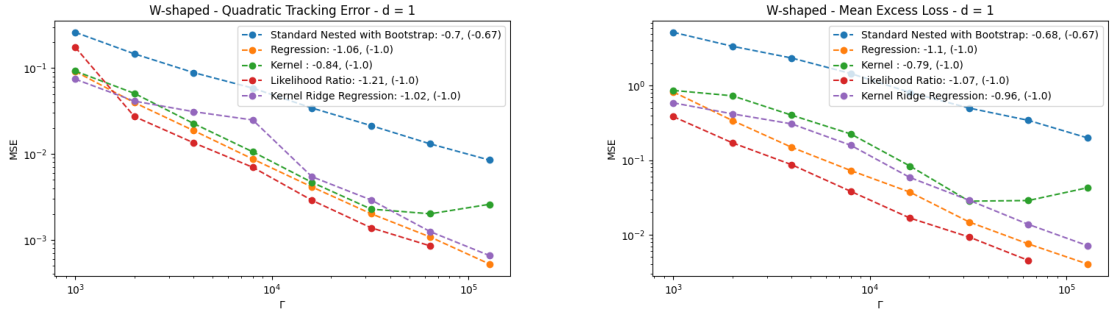


Figure 2.8: Empirical convergence of nested simulation procedures for a W-shaped payoff

In Broadie et al. (2015), the authors propose a numerical example where the payoff has a W-shape with respect to the asset price at maturity. We have incorporated this example as Portfolio 5. In Figure 2.8, the empirical convergence of the nested simulation procedures for a quadratic tracking error and a mean excess loss on Portfolio 5 is illustrated for $d = 1$. For all procedures, we observe a similar convergence behavior as in other path-dependent options. The MSEs of the kernel smoothing-based and VaR-based procedures do not decrease monotonically as the simulation budget increases. This fluctuating behavior is observed and analyzed in Section 2.5.2.

Figure 2.9 illustrates similar observations for sensitivity to different risk measures. A change in the risk measure of interest does not affect the empirical convergence behavior of

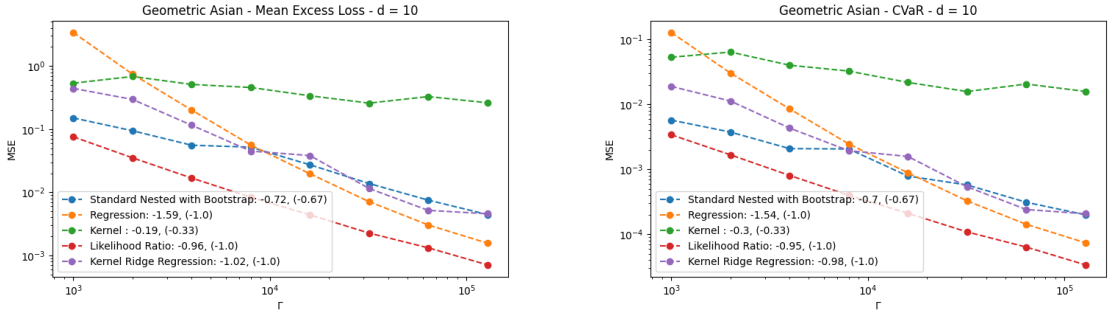


Figure 2.9: Empirical convergence of nested simulation procedures for different risk measures on Portfolio 1 with $d = 20$

all nested simulation procedures. From the empirical convergence results, we observe that the empirical rates of convergence of the regression-based nested simulation procedure are the highest among all nested simulation procedures for all risk measures and option types. Furthermore, the regression-based procedure is stable across different payoff structures and risk measures. Its MSEs decrease quickly in the beginning, and after reaching a certain level of Γ, the rate of decrease stabilizes to match its asymptotic rate of convergence. Parametric regression is the most attractive metamodel for nested simulation procedures due to its fast empirical convergence behavior, stability across different payoff structures and risk measures, and ease of implementation. The following sections focus on the regression-based nested simulation procedure for the rest of the numerical experiments.

2.5.5 Sensitivity to level for VaR and CVaR

The level of VaR and CVaR is a critical factor that determines the complexity of the risk measure estimation. In our previous numerical experiments, the level of VaR and CVaR is set to be the 90% quantile of the distribution of the inner simulation noise. In this section, we examine the empirical convergence of the regression-based nested simulation procedures for other levels of VaR and CVaR, i.e., 80%, 95%, 99%, and 99.6%.

In Figure 2.10, the empirical convergence of the regression-based nested simulation procedure for different levels of VaR and CVaR is illustrated for up-and-out barrier call options. The empirical rates of convergence of the regression-based nested simulation procedure are not sensitive to the level of VaR and CVaR.

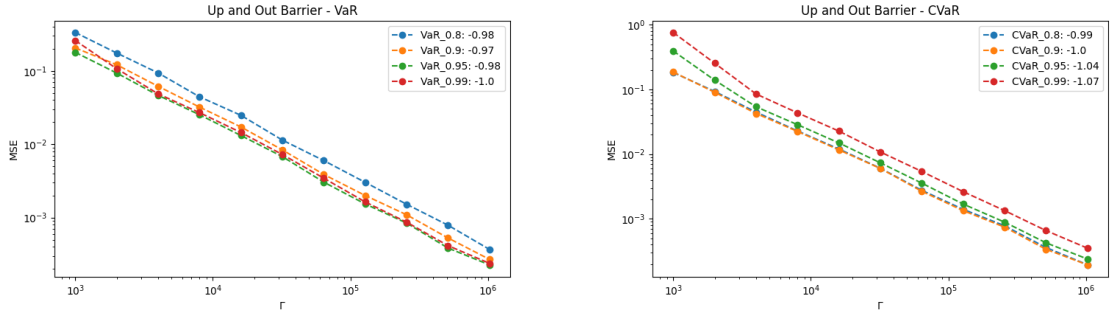


Figure 2.10: Empirical convergence of regression-based procedures for different levels of VaR and CVaR for Up and Out Barrier Call Options

2.5.6 Sensitivity to the Asset Model

Our previous numerical experiments have been conducted under the assumption that the underlying asset dynamics follow a multidimensional geometric Brownian motion with 0.3 pairwise correlation. To examine the sensitivity of the empirical convergence behavior to the asset model, we consider a stochastic volatility model of a Heston type (Heston, 1993). In a Heston model, the asset price and the volatility are correlated, and the volatility is a stochastic process.

$$
\begin{aligned}
dS_t &= \mu S_t dt + \sqrt{v_t} S_t dW_{1,t}, \\
dv_t &= \kappa(\theta - v_t)dt + \xi\sqrt{v_t} dW_{2,t},
\end{aligned}
$$

where $W_{1,t}$ and $W_{2,t}$ are two independent Brownian motions.

In this experiment, the initial asset price and the volatility are set to be 100 and 0.2, respectively. The parameters are set to be $\mu = 0.08$, $\kappa = 2$, $\theta = 0.04$, $\xi = 0.1$.

In Figure 2.11, the empirical convergence results of the regression-based nested simulation procedure for a quadratic tracking error and a 90%-CVaR are illustrated for different asset models, i.e., a geometric Brownian motion and a Heston model. The empirical rates of convergence of the regression-based nested simulation procedure are observed to be insensitive to the asset model.

In summary, we find the empirical convergence behavior of the regression-based nested simulation procedure to be stable across different risk measures, option types, asset dimensions, levels of VaR and CVaR, asset models, and regression bases. With a limited total

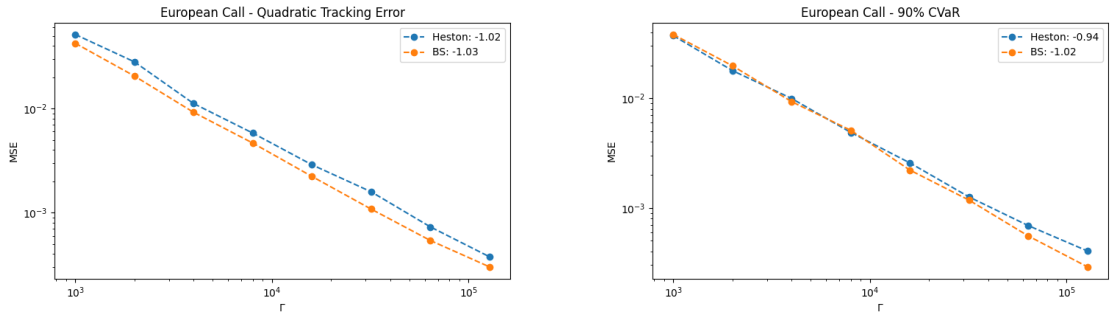


Figure 2.11: Empirical convergence of regression-based nested simulation procedures for different asset models

budget, the regression-based nested simulation procedure is the most robust and stable among all nested simulation procedures.

2.5.7 Empirical Convergence of MLMC

The MLMC procedure is a variance reduction technique that is designed to reduce the computational cost of nested simulation procedures. The theoretical analysis in Giles and Haji-Ali (2019) shows that the MLMC procedure has a faster empirical rate of convergence than the standard nested simulation procedure when the risk measure of interest is the probability of a large loss over a threshold u, i.e., the nested expectation case with h being an indicator function. In this numerical experiment, the initial number of inner replications, the refinement factor, the r-value for adaptive algorithm are set to be 4, 4, and 1.5, respectively. We note that the MLMC procedure focuses on adaptive allocation of the simulation budget and is fundamentally different from supervised learning-based nested simulation procedures. And their results are not directly comparable.

We provide a summary of the empirical convergence results of the MLMC procedure with a decomposition table of the MSEs in Table 2.4. The MSEs of the MLMC procedure are decomposed into the bias and the variance of the estimator. The MSEs of the standard nested simulation procedure with a similar total simulation budget are also provided for comparison. Since the MLMC procedure is designed with a fixed number of outer simulation paths, the benefit of the MLMC procedure is minimal when the total simulation budget is large. More specifically, when Γ is large, the standard nested simulation procedure with a bootstrap-based budget allocation strategy benefits from having a larger number of outer simulation paths, and the MSE of the standard nested simulation proce-

Level	Bias	Variance	MSE	Γ	MSE of SNS
0	0.118	0.104	0.1364	6400	0.1660
1	0.102	0.0916	0.1020	27600	0.1305
2	0.0870	0.0794	0.0870	76600	0.0954
3	0.0815	0.0746	0.0812	175600	0.0744
4	0.0893	0.0805	0.0884	348100	0.0460
5	0.0750	0.0694	0.0750	640000	0.0366

Table 2.4: MSEs of the MLMC procedure for different levels

dure is lower than that of the MLMC procedure with a fixed number of outer simulation paths.

2.6 Computational Complexity

Compared to the standard procedure, metamodel-based nested simulation procedures often have faster empirical rates of convergence. Their faster convergence benefits from the pooling of inner samples through the metamodels. However, pooling itself comes with increased computational costs, which are usually ignored in most numerical comparisons. This section summarizes the algorithmic complexity and illustrate the computational time of different nested simulation procedures. The findings can provide some further guidance on the choice of a proper nested simulation procedure given a nested estimation problem. Define basic operations to be basic mathematical operators, that is, arithmetic with individual elements has complexity $\mathcal{O}(1)$.

Procedures	Training cost	Prediction cost	Total additional cost
Standard procedure	0	$\mathcal{O}(M)$	$\mathcal{O}(M)$
Regression	$\mathcal{O}(p^2M) + \mathcal{O}(p^3)$	$\mathcal{O}(pM)$	$\mathcal{O}(p^2M) + \mathcal{O}(p^3)$
Kernel smoothing	$\mathcal{O}(M\log(M))$	$\mathcal{O}(k\log(M))$	$\mathcal{O}(M\log(M))$
Kernel ridge regression	$\mathcal{O}(M^3)$	$\mathcal{O}(M^2)$	$\mathcal{O}(M^3)$
Likelihood ratio	0	$\mathcal{O}(M^2)$	$\mathcal{O}(M^2)$

Table 2.5: Additional computational costs of nested simulation procedures aside from simulation

Table 2.5 shows the algorithmic complexity of the nested simulation procedures. For a d-dimensional nested estimation problem, the simulation cost for all procedures is $\mathcal{O}(dMN)$.

The simulation cost is omitted from Table 2.5, as all procedures are given the same simulation budget. Given M outer samples and the corresponding M inner sample averages, the algorithmic complexity of the additional operations does not involve N and thus, can be written in terms of M only. The training cost includes the cost of fitting the metamodel and the cost of cross-validation for non-parametric regression metamodels.

For the standard procedure,

- the training cost is 0 due to the absence of a metamodel.
- Its prediction cost $\mathcal{O}(M)$ comes from the evaluation of the function h for M samples.

For the regression-based procedure, a linear regression model is fitted with p predictors.

- The training cost is $\mathcal{O}(p^2M) + \mathcal{O}(p^3)$. Estimating the coefficients of the regression model involves multiplying the design matrix by its transpose and inverting the resulting matrix, which has complexity $\mathcal{O}(p^2M)$ and $\mathcal{O}(p^3)$, respectively. For a review in the complexity of a matrix inversion, see Stothers (2010). The matrix inversion has complexity $\mathcal{O}(p^3)$ using a Gauss-Jordan elimination, but in practice, the complexity can be reduced to $\mathcal{O}(p^{2.807})$ by Strassen (1969) and $\mathcal{O}(p^{2.376})$ by Coppersmith and Winograd (1987). In practice, p is usually much smaller than M and does not grow with the simulation budget Γ. p is fixed by the basis function, which is selected based on the complexity of the payoff function.
- The prediction cost of the regression-based procedure is $\mathcal{O}(pM)$, which comes from the multiplication of the design matrix by the estimated coefficients.

For the kernel smoothing-based procedure, the kNN metamodel is implemented. In practice, the K-D tree algorithm (Bentley, 1975) is often implemented for an efficient nearest neighbor search.

- During training, a tree is constructed to store the distance, which has a degree of complexity of $\mathcal{O}(M \log(M))$.
- The prediction cost of the kernel smoothing-based procedure is $\mathcal{O}(kM \log(M))$, which comes from a query of the K-D tree for k nearest neighbors for M samples. Conversely, an inefficient algorithm calculates the distance matrix during training and a block sort algorithm to find the k nearest neighbors, which results in a degree of complexity of $\mathcal{O}(M^2)$ and $\mathcal{O}(M^2 \log(M))$, respectively. A practical implementation of a block sort is provided in Kim and Kutzner (2008). Hence, an efficient implementation of the kNN metamodel is crucial for the kernel smoothing-based procedure.

For the likelihood ratio-based procedure,

- the training cost is 0 as no training is required.
- The prediction cost of the likelihood ratio-based procedure is $\mathcal{O}(M^2)$, which comes from the calculation of the likelihood weights for M samples.

For the VaR-based procedure,

- the main training cost comes from inverting an $M \times M$ kernel matrix (Schölkopf and Smola, 2002), which has a degree of complexity of $\mathcal{O}(M^3)$ using a Gauss-Jordan elimination. Similar to the regression-based procedure, this complexity can be reduced to $\mathcal{O}(M^{2.376})$.
- The prediction cost of the VaR-based procedure is $\mathcal{O}(M^2)$, which comes from the multiplication of the kernel matrix by the estimated coefficients.

Among the metamodel-based procedures, parametric regression is the most efficient. In the multiple linear regression, the number of features is usually much less than the number of data points, i.e., $p < M$. Kernel smoothing and VaR are kernel-based metamodels, and they are more computaional expensive due to distance calculations and cross-validations of hyperparameters. The kNN metamodel has only 1 hyperparameter k, while the VaR metamodel requires 3 hyperparameters, namely a smoothness hyperparameter ν, a scale hyperparameter υ, and a regularization hyperparameter λ. Calculating the likelihood ratio weights is inevitable for the likelihood ratio-based procedure. While a k-fold cross-validation is used to estimate the hyperparameter for kNN, the VaR requires Bayesian optimization (Shahriari et al., 2015) to estimate the hyperparameters due to having a high-dimensional search space. The likelihood ratio-based procedure requires no training, but the cost of calculating the likelihood weights is $\mathcal{O}(M^2)$. Comparing the algorithmic complexity of the nested simulation procedures, the regression-based procedure is the most efficient among the metamodel-based procedures. For a fixed p, the total additional cost of a regression-based procedure is $\mathcal{O}(M)$, which is the same as the standard procedure's.

The cost associated with pooling makes a major impact on the overall computational complexity of a metamodel-based nested simulation procedure. In our finite-sample experiments, we have indeed observed that the actual computational cost to deviate substantially from the simulation cost. In Figure 2.12, the total actual computational time of different nested simulation procedures for European call options and geometric Asian options with $d = 10$ is illustrated in a log-log scale. The x-axis shows the total simulation budget Γ,

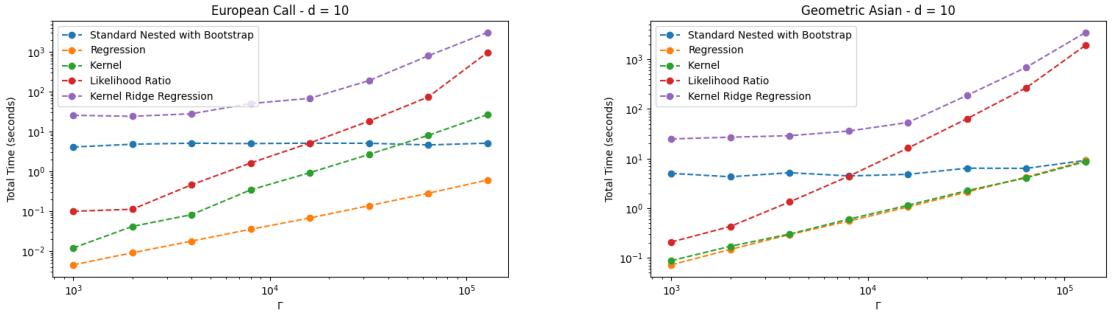


Figure 2.12: Total computational cost for different procedures with $d = 10$

and the y-axis is the total computational time for 1 replication of the numerical experiment, in seconds. All procedures are implemented on a machine with an AMD Ryzen 9 7900X processor with 32 GB of RAM. 8 cores are used for parallel computing, and the total computational time is the sum of the simulation time, the training time, and the prediction time. The regression and kernel smoothing-based procedures are the most efficient among the metamodel-based procedures. Since the regression-based procedure has a higher empirical rate of convergence, it is preferred over kNN. The likelihood ratio-based and VaR-based procedures are the most computationally expensive among all procedures. They are also demanding in terms of memory. For budgets larger than 10^5, the likelihood ratio-based procedure becomes impractical as storage of the likelihood weights becomes a bottleneck. The VaR-based procedure suffers from both cross-validation and inverting a kernel matrix of size $M \times M$. To separate the total computational time into different attributes, we provide a detailed analysis of the total computational time in the remainder of this section.

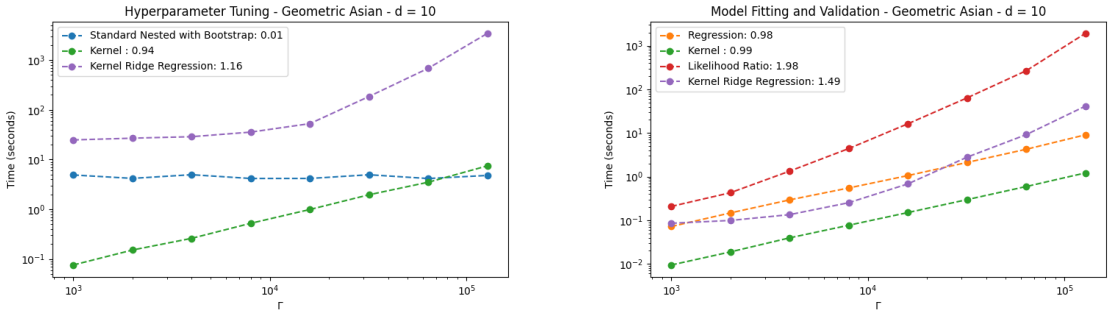


Figure 2.13: Computational cost for implementing nested simulation procedures with $d = 10$, excluding simulation time

Figure 2.13 illustrates the computational time of implementing different nested simulation procedures for the portfolio of geometric Asian options with $d = 10$. The simulation time is omitted as it is the same for all procedures. The remaining computations can be decomposed into two parts: hyperparameter tuning and model implementation. The hyperparameter tuning cost is the cost of estimating the optimal hyperparameters for the metamodels, and the model implementation cost is the cost of fitting the metamodels and generating predictions from the trained metamodels. For each procedure, each point in Figure 2.13 represents the average computational time for a given simulation budget Γ. A regression model is fitted to each model respectively, and its slope can be viewed as the growth rate of the computational time with respect to Γ.

- The total computational time of the standard procedure is mostly attributed to finding the optimal M and N using the bootstrap-based budget allocation strategy from Zhang et al. (2021). This cost does not grow with the simulation budget Γ, as its slope is close to 0.

- The regression-based procedure does not require any hyperparameter tuning, and its total computational time is mainly attributed to the model implementation. The slope of the regression line is close to 1, resembling its algorithmic complexity reported in Table 2.5.

- The kernel smoothing-based and VaR-based procedure requires a cross-validation procedure to estimate the optimal hyperparameters. In our implementation of kNN, the cross-validation is conducted by using a grid search with a search space which cardinality does not grow with the simulation budget Γ. However, the cross-validation cost of the kernel smoothing-based procedure grows with the simulation budget Γ as more samples are involved. The VaR-based procedure requires a Bayesian optimization to estimate the optimal hyperparameters, and the cardinality of the search space also grows with the simulation budget Γ. The empirical cost of the VaR-based procedure does not resemble its algorithmic complexity reported in Table 2.5. The efficient implementation of VaR is an active area of research, and the computational cost of VaR can be reduced by using a low-rank approximation of the kernel matrix, e.g., the Nyström method (Nyström, 1930). Our observation for the VaR implementation time is in line with the findings in Pedregosa et al. (2011), where the cost of VaR is reported to be increasing quickly with the number of samples.

- The likelihood ratio-based procedure requires no hyperparameter tuning, and its total computational time is mainly attributed to the likelihood weight calculation, which is $\mathcal{O}(M^2)$.

In summary, for a large simulation budget, the regression-based procedure is the most efficient among all metamodel-based procedures in terms of computational time. For moderate simulation budgets, the regression-based procedure is a preferred choice.

2.7 Conclusion and Extensions

In the task of estimating risk measures for portfolios of complex financial derivatives, nested simulation procedures are commonly required but often computationally expensive. Tremendous efforts have been made to improve the efficiency of nested simulation procedures by approximating the inner simulation model with a metamodel. In this study, we review the literature on nested simulation procedures in financial engineering and provide fair comparisons of different metamodels by using the same set of numerical examples. Asymptotic properties of estimators for different nested simulation procedures are influenced by their corresponding metamodels. To show an asymptotic convergence result, a more complex metamodels requires a more stringent set of assumptions on the distribution of the outer scenario and the inner simulation noise. With extensive numerical experiments, we have found that the finite-sample performance of a procedure can deviate from its theory. In theory, supervised learning-based nested simulation procedures often provide higher rates of convergence, but they come at the computational expense of model training and generating predictions from the trained models. The likelihood ratio-based procedure requires no training, but it is expensive to compute, and it is also expensive to store the likelihood weights. As a result, the total computational budget is not necessarily the same as the simulation budget, which is usually a limiting factor for practical applications. A kernel-based procedure, e.g., kNN and VaR, requires cross-validation, and its empirical performance depends heavily on the choice of hyperparameters. A kNN-based procedure is sensitive to the asset dimension and the problem complexity. A VaR-based procedure is computationally expensive, and its cost grows quickly with the simulation budget. For kernel-based procedures, the computational cost is heavily dependent on the efficient implementation of the associated algorithms. For a nested estimation problem with a given computational budget, we suggest the use of the regression-based simulation procedure when the budget size is moderate. It is efficient to implement, and it exhibits fast empirical convergence in estimating risk measures for popular option portfolios. However, we have only examined the performance for option portfolios in this study, where finding a suitable set of basis functions for regression is relatively easy. In practice, a variable annuity contract is often high-dimensional in time with a complex payoff structure, and it is a challenging problem for nested simulation procedures. The time dependence in the

feature space leads to multi-collinearity, and the regression-based procedure may not be the best choice in such instances. In such cases, a neural network-based procedure may be more suitable, as it is a data-driven algorithm that finds a suitable set of basis functions automatically. In Chapter 3, we examine the performance of regression-based and neural network-based nested simulation procedures for estimating risk measures for variable annuities.

Chapter 3

Cutting Through the Noise: Using Deep Neural Network Metamodels for High Dimensional Nested Simulation

3.1 Introduction

DNN have attracted attentions of researchers and practitioners due to their success (Hastie et al., 2009; LeCun et al., 2015) in solving real-world machine learning tasks such as AlphaGo (Silver et al., 2016) and ChatGPT (OpenAI, 2023). Since the first artificial neural network model (McCulloch and Pitts, 1943) and the first algorithm for training a perceptron (Rosenblatt, 1958), especially after the introduction of backpropagation (Rumelhart et al., 1985) and the growth of high-performance computing, the field of artificial neural network and deep learning in general has been growing rapidly. Two specialized neural network architectures that are relevant to our study are RNN (Williams and Zipser, 1989; Sutskever et al., 2014) and LSTM (Hochreiter and Schmidhuber, 1997; Chung et al., 2014). Despite their success, DNN models are often criticized for their lack of transparency and interpretability, which hinders their adoption in financial and actuarial applications. Enormous research efforts have been spent to test and improve the robustness of DNN models with carefully designed noise injection methods. Poole et al. (2014) show that injecting synthetic noise before and after hidden unit activations during training improves the performance of autoencoders. Neelakantan et al. (2015) improve learning for DNNs by injecting

synthetic noise to the gradients during backpropagations. A branch of research has been devoted to understanding the resilience of neural network models to noise in training labels. For example, Luo et al. (2016) show that adding synthetic label noise to the convolutional neural network (Convolutional Neural Network (CNN)) can improve its ability to capture global features. Srivastava et al. (2014) quantify the error tolerance by injecting synthetic label noise with a custom Boltzmann machine hardware. Szegedy et al. (2013) find that neural networks are vulnerable to adversarial examples, and Goodfellow et al. (2014) design an efficient method to generate such noisy examples to exploit the vulnerability to adversarial perturbations. Carlini and Wagner (2017) design targeted attacks to training labels to test the robustness of neural networks. Instead of using synthetic noise, Jiang et al. (2020) inject real-world label noise and examine noise tolerance of neural networks with controlled noise levels. The aforementioned studies use real-world data, as is typically the case for many neural network studies, where noise is already present in the training labels before any noise injection. Users of real-world data have little control over the noise level of the original training labels and usually examine the effect of noisy data by injecting noise, but it is unclear whether a neural network model trained on noisy data actually learns the real, i.e., noiseless, feature-label relationship. Due to their lack of transparency and interpretability, the adoption of DNNs in financial and actuarial applications has been received by regulators with some skepticism.

The contributions of our study in this chapter are two-fold:

1. We study what DNNs learn from noisy data by training them using simulated data based on well-designed simulation experiments. This is a novel way to study the effect of noisy data and error tolerance of neural network models as one can *reduce noise* in the data by increasing the number of replications in a simulation model. This new way of studying neural network models can provide more direct evidence on their transparency and interpretability.

2. We propose two generic nested simulation procedures that uses DNNs as metamodels to improve its efficiency while maintaining transparency. In essence, a pilot stage simulation is used to generate a large number of noisy data, which are then used to train a metamodel. Depending on the application, a trained metamodel can serve two purposes:

   - to identify a set of tail scenarios, and
   - to estimate risk measures directly.

   The first procedure uses a metamodel to identify a set of potential tail scenarios on which computations are performed in stage 2, while the second procedure uses

metamodel predictions to estimate risk measures directly. Our numerical results show that DNN metamodels can identify the tail scenarios accurately and so the proposed procedures can estimate tail risk measures with a similar degree of accuracy while, at the same time, using less simulation budget.

In essence, we are curious about two fundamental questions:

1. "What do DNNs learn from noisy data?" and
2. "How well do neural networks learn from noisy data?".

Data-driven answers to these questions prevail in the existing literature. In supervised learning, DNNs are believed to learn from a given data about the feature-label relationship to predict new labels for unseen features. A cross-validation procedure used to assess a subset, i.e., the validation set, of the original data, is a common way to access the quality of learning. Generalization error on the test labels is another popular assessment metric. However, the test set is also a subset of the original data. In this study, we revisit these questions in a simulation context and propose an alternative approach to answer them. Instead of relying solely on real-data and splitting it into multiple subsets, we propose using stochastic simulation outputs as training labels for DNNs. By controlling the simulation design parameters, such as the number of independent replications, we can control the quality and also the quantity of the training labels fed into the neural networks. In such a controlled environment, we obtain more clear-cut answers to the above fundamental questions.

Estimating tail risk measures of portfolios of complex VAs usually requires nested simulation. In the nested simulation, the outer simulation involves projecting scenarios of key risk factors under a $\mathbb{P}$ measure, while the inner simulations are used to value payoffs uder guarantees of varying complexity, under a $\mathbb{Q}$ measure.

For insurers, large-scale nested simulations for assessing VA losses are generally computationally intensive; i.e., a large number of outer level simulations are needed to estimate extreme tails for calculations of tail risk measures, and a large number of inner simulations are also needed at each time step, because the embedded options are out of money. In addition, the required calculations need to be repeated for each VA contract, or cluster of VA contracts. Consequently, insurers are interested in techniques for nested simulation models that produce accurate results within the confines of a finite computational budget.

There are two main approaches that can be used to improve the efficiency of inner simulations. The first one uses a dynamic, non-uniform allocation of the inner simulation

budget, and the second uses a proxy model to replace the inner simulation step. Some work uses a combination of both approaches.

In Chapter 2, the likelihood ratio approach is used to allocate the inner simulation budget using likelihood ratio weights, i.e., importance sampling. Proxy models are tractable analytic functions or flexible empirical functions that are used to replace the inner simulation stage of a nested simulation. Empirical proxies may be intrinsic or extrinsic to the simulation process. Extrinsic proxy functions are selected to be close to the inner simulation values and therefore require detailed information about the payoffs that are evaluated in the inner simulation step. Empirical proxies are constructed by using an initial pilot simulation to develop factors or functionals that can subsequently be used in place of the inner simulation. A prominent expirical proxy method discussed in the literature is the least squares Monte Carlo method (Longstaff and Schwartz, 2001). In Chapter 2, supervised learning models, including linear regression, kNN, and KRR, are used as proxy models for inner simulations. This chapter introduces a proxy model to reduce the computational requirements of inner simulations of VAs by replacing the inner simulation step of a nested simulation procedure. This proxy model is chosen to be a flexible empirical function, i.e., a DNN metamodel.

In our study, supervised learning metamodels are trained using data simulated from an initial pilot simulation. They are then used to replace the time-consuming inner simulations by the trained model. Borrowing terminologies from the machine learning community, we can view a set of simulated outer scenarios and estimated hedging errors for those scenarios as *features* and noisy *labels*. The noise level of the labels can be controlled by the number of inner replications in the inner simulation model. We refer to the trained supervised learning proxy models as *metamodels* of the inner simulation, which is also known as the *surrogate models* in the simulation literature. Metamodeling is a popular approach to reduce the computational burden of simulation-based applications by replacing the time-consuming simulation with a metamodel. The metamodel is trained using a set of simulated data, and it is used to predict the simulation output for new inputs. The study of metamodeling is an active research area in the simulation literature, and using DNNs as metamodels is a relatively new development. Fonseca et al. (2003) provide general guidelines for simulation metamodeling with neural networks, Lieu et al. (2022) use DNNs as metamodels of a simulation model for a structural reliability analysis, and Salle and Yıldızoğlu (2014) show that neural network metamodels help achieve higher degree of prediction accuracy that other metamodels in approximating agent-based simulation models. A popular metamodel in nested simulation procedures is stochastic kriging. Liu and Staum (2010) use stochastic kriging as a metamodel of Monte Carlo simulations to estimate the CVaR of a portfolio of derivative securities, and Gan and Lin (2015) use

a stochastic kriging for an efficient valuation of large portfolios of VA contracts. Other studies, such as Broadie et al. (2015), Hong et al. (2017), and Zhang et al. (2022) use a regression, a kernel smoothing, and a likelihood ratio method, respectively as metamodels. Our study has three key distinctions over the existing ones:

1. our metamodel has high-dimensional inputs. In machine learning terminology, the features are high-dimensional vectors. To estimate the hedging error of a typical VA contract, the number of features is in the order of hundreds, which is at least one order of magnitude larger than the number of features in the aforementioned studies,

2. for estimating tail risk measures, our metamodel is only used for tail scenario identification but is *not* used in the estimation of the tail risk measures. This is a feature designed particularly to convince regulators that the losses used in estimating the risk measure are based on a transparent inner simulation model rather than on some black-box metamodels, and

3. using simulation models as data generators, we can decrease the noise level and get arbitrarily close to the true labels by increasing the number of replications in the simulation model. This design allows a systematic study of the effect of noisy training labels on the performance of DNN models in predicting the noiseless labels.

In this chapter, DNN metamodels and DNN models are used almost interchangeably despite one distinction.

- The discussion of *DNN metamodels* focuses more on the aspect of *estimating* the inner simulation.

- The discussion of *DNN models* focuses more on *studying the DNN* using simulation data.

The rest of this chapter is organized as follows. Section 3.2 presents the problem settings for tail risk measures and dynamic hedging of VAs. Section 3.3 proposes an efficient two-stage nested simulation procedure that uses DNNs as metamodels to help reduce a simulation budget by only performing computations on identified tail scenarios. Section 3.4 proposes a single-stage nested simulation procedure that estimates risk measures directly with metamodel predictions. Section 3.5 demonstrates the efficiency of DNN metamodels and examines the error tolerance of two LSTM metamodels with different numbers of trainable parameters. Lastly, practical suggestions are provided for the choice of suitable metamodels and simulation settings.

3.2 Problem Formulation

In this section we present notations, problem settings, and a simulation model for risk estimation for hedging errors of VAs. A main goal of this section is to showcase the complexity of the simulation model, which we use as a data generator to train DNN metamodels (Section 3.3 and Section 3.4). For readers who are interested in the examination of a neural network metamodel, it is sufficient to understand that our simulation model generates data with 240 features and 1 real-value label and our metamodels are generally applicable to any simulation model that generates data with similar characteristics.

3.2.1 Tail Risk Measures

Measuring and monitoring risks, particularly tail risks, are important risk management tasks for financial institutions such as banks and insurance companies. Two most popular tail risk measures are VaR and CVaR (Hardy and Saunders, 2022; Rockafellar and Uryasev, 2002).

Consider a loss random variable L which losses and gains lie in the right and left tails, respectively, of its distribution. For a given confidence level $\alpha \in [0,1]$, the α-VaR and α-CVaR are defined in Equation (2.1) and Equation (2.2), respectively. Tail risk measures such as VaR and CVaR are widely used for setting a regulatory and economic capital, which is the amount of capital a financial institution holds to cover its risk. For example, European insurers set regulatory capital at 99.5%-VaR according to Solvency II EIOPA (2014). In Canada, the regulatory capital requirement for VAs is set based on CVaRs as prescribed in OSFI (2017).

Let $L_1, L_2, \ldots, L_M$ be M i.i.d. simulated losses of L and let $L_{(1)} \leq L_{(2)} \leq \ldots \leq L_{(M)}$ be the ordered losses. For a given confidence level α (assume that αM is an integer for simplicity), α-VaR can be estimated by the sample quantile $\widehat{\text{VaR}}_\alpha = L_{(\alpha M)}$. Also, α-CVaR can be estimated by

$$\widehat{\text{CVaR}}_\alpha = \frac{1}{(1-\alpha)M} \sum_{i=\alpha M+1}^{M} L_{(i)} = \frac{1}{(1-\alpha)M} \sum_{i \in \mathcal{T}_{(1-\alpha)M}} L_i. \tag{3.1}$$

Equation (3.1) is the same as (2.3) except that we define a *true tail scenario set* of size k as $\mathcal{T}_k = \{i : L_i > L_{(M-k)}\}$. In this study, the loss random variable of interest is the hedging error for VA.

3.2.2 Simulation Model for Variable Annuity Payouts

VA contracts offer different types of guarantees. Generally speaking, a portion of the VA premium is invested in a sub-account which return is linked to some stock indices.

Two relevant types of guarantees in our studies are:

- **GMMB**: A GMMB contract pays a maturity benefit equal to the greater of the sub-account value and a fixed guarantee value. The guarantee value is often set as a percentage, e.g., 75% or 100%, of the initial premium.
- **GMWB**: A GMWB contract guarantees the minimum amount of periodic withdrawal the policyholder can take from the sub-account until maturity, even if the sub-account value reduces to zero. The minimum withdrawal benefit is typically a fixed percentage of the guarantee value. The guarantee value will decrease if the withdrawal exceeds the guaranteed minimum. The GMWB is typically offered with an accumulation period, during which no withdrawals are made but a Guaranteed Minimum Accumulation Benefit (GMAB) is usually offered. Additional features offered with the GMWB include roll-up, ratchet, and reset (The Geneva Association, 2013).

For a comprehensive review of other types of VA contracts such as GMAB, Guaranteed Minimum Death Benefit (GMDB), and Guaranteed Lifetime Withdrawal Benefit (GLWB), we refer readers to Hardy (2003). Next we present a summary of dynamic hedging for VA contracts. We refer readers to Dang (2021) for detailed modeling of insurer liabilities in different VA contracts and Greek estimation.

Consider a generic VA contract with maturity $T > 0$ periods, e.g., $T = 240$ months. Denote the policyholder's (random) time of death by $\tau > 0$. The contract expires at $T' = \min\{T, \tau\}$, which isthe earlier of the contract maturity and the death of the policyholder. Let S_t, F_t, and G_t be an indexed stock price, the subaccount value and a guarantee value, respectively, at time $t = 1, 2, \ldots, T$. Evolution of the subaccount value and the guarantee value of a VA contract affect the contract payout. Let the triplet $X_t = (S_t, F_t, G_t)$ be the state of the VA contract at time t. Note that the policyholder's (random) time of death also affects the timing of the benefit payout for certain types of VA such as GMAB, but this is not considered in our study for simplicity. For clarity, we use F_t and F_{t_+} to denote the sub-account value just before and just after the withdrawal at time t, if any. Let η_g be a gross rate of management fee that is deducted from the fund value at each period and let $\eta_n < \eta_g$ be a net rate of management fee income to the insurer. The difference between the gross management fee and the net management fee income represents incurred investment expenses.

At the inception of the contract, i.e., $t = 0$, we assume that the whole premium is invested in the stock index and the guarantee base is set to the sub-account value:

$$S_0 = F_0 = G_0.$$

At each time $t = 1, \ldots, T$, the following events take place in order:

1. The sub-account value changes according to the growth of the underlying stock and the (gross) management fee is deducted, which is,

$$F_t = F_{(t-1)_+} \cdot \frac{S_t}{S_{t-1}} \cdot (1 - \eta_g),$$

   where $(x)^+ = \max\{x, 0\}$ and $F_{(t-1)_+}$ will be defined later. The insurer's income at time t is a net management fee, i.e., $F_t \eta_n$.

2. The guarantee value ratchets up (ratcheting is a common feature in GMWB) if the sub-account value exceeds the previous guarantee value,

$$G_t = \max\{G_{t-1}, F_t\}.$$

3. The withdrawal is made (for GMWB) and is deducted from the sub-account value,

$$F_{t_+} = (F_t - I_t)^+,$$

   where $I_t = \gamma G_t$. A GMMB can be modeled with $\gamma = 0$.

We see from the above modeling steps that the status of a generic VA contract is summarized by a triplet (S_t, F_t, G_t) which evolution is driven by the stochasticity of S_t. In practice, the simulation model may also incorporate additional complications such as mortality, lapse, and excess withdrawal, etc.

At any time $t = 1, \ldots, T$, the insurer's liability in a VA contract is the present value of all payments, net of the fee income. For example, suppose that the per-period risk-free rate is r, then the insurer's time-t liability for a GMWB contract is

$$V_t = V(\mathbf{S}_t) = \mathbb{E}\left[e^{-r(T-t)} \cdot (G_T - F_T)^+ - \sum_{s=t+1}^{T} e^{-r(T-s)} F_s \eta_n \Big| \mathbf{S}_t \right], \tag{3.2}$$

where $\mathbf{S}_t = (S_0, S_1, \ldots, S_t)$ is the (outer) sample path up to time t, and the conditional expectation is taken with respect to the inner sample paths $\widetilde{S}_{t+1}, \ldots, \widetilde{S}_T$ given the outer

path $\mathbf{S}_t$. The tilde symbol ($\sim$) over a quantity denotes its association with the inner simulation. In practice, given the stock sample path, which is an outer path $S_1, \ldots, S_t$, one can simulate these future stock prices $\widetilde{S}_{t+1}, \ldots, \widetilde{S}_T$ with an inner simulation model that is based on some asset model such as Geometric Brownian Motion (GBM) or RS-GBM. Given the time t state (S_t, F_t, G_t), following Cathcart et al. (2015) the sensitivity of V_t with respect to S_t can be estimated by a pathwise estimator (Glasserman, 2004):

$$\begin{aligned}
\Delta_t(\widetilde{S}_{t+1}, \ldots, \widetilde{S}_T | \mathbf{S}_t) &= \frac{\partial V_t}{\partial S_t} \\
&= \frac{\partial}{\partial S_t} \mathbb{E}\left[e^{-r(T-t)} \cdot (G_T - F_T)^+ - \sum_{s=t+1}^{T} e^{-r(T-s)} F_s \eta_n \middle| \mathbf{S}_t \right] \\
&= \mathbb{E}\left[\frac{\partial}{\partial S_t} \left(e^{-r(T-t)} \cdot (G_T - F_T)^+ - \sum_{s=t+1}^{T} e^{-r(T-s)} F_s \eta_n \right) \middle| \mathbf{S}_t \right] \qquad (3.3)
\end{aligned}$$

For an inner simulation path $\widetilde{S}_{t+1}, \ldots, \widetilde{S}_T$ of the GMWB contract, the sensitivity of V_t can be calculated by the following recursion:

$$\sum_{s=t+1}^{T} e^{-r(T-s)} \left[\mathbf{1}\{\widetilde{I}_{t,s} > \widetilde{F}_{t,s}\} \cdot \left(\frac{\partial \widetilde{I}_{t,s}}{\partial S_t} - \frac{\partial \widetilde{F}_{t,s}}{\partial S_t} \right) - \eta_n \frac{\partial \widetilde{F}_{t,s}}{\partial S_t} \right], t = 0, \ldots, T-1,$$

where $\mathbf{1}\{\cdot\}$ is an indicator function, $\widetilde{S}_t = (S_{t,t+1}, \ldots, S_{t,T})$, $\widetilde{F}_t = (F_{t,t+1}, \ldots, F_{t,T})$, $\widetilde{G}_t = (G_{t,t+1}, \ldots, G_{t,T})$, $\widetilde{I}_t = (I_{t,t+1}, \ldots, I_{t,T})$, and

$$\begin{aligned}
\frac{\partial \widetilde{F}_{t,s}}{\partial S_t} &= \mathbf{1}\{\widetilde{I}_{t,s-1} < \widetilde{F}_{t,s-1}\} \cdot \left(\frac{\partial \widetilde{F}_{t,s-1}}{\partial S_t} - \frac{\partial \widetilde{I}_{t,s-1}}{\partial S_t} \right) \cdot \frac{\widetilde{S}_{t,s}}{\widetilde{S}_{t,s-1}} \cdot (1 - \eta_g), \\
\frac{\partial \widetilde{G}_{t,s}}{\partial S_t} &= \mathbf{1}\{\widetilde{G}_{t,s-1} < \widetilde{F}_{t,s}\} \cdot \frac{\partial \widetilde{F}_{t,s}}{\partial S_t} + \mathbf{1}\{\widetilde{G}_{t,s-1} \geq \widetilde{F}_{t,s}\} \cdot \frac{\partial \widetilde{G}_{t,s-1}}{\partial S_t}, \\
\frac{\partial \widetilde{I}_{t,s}}{\partial S_t} &= \gamma \frac{\partial \widetilde{G}_{t,s}}{\partial S_t}. \qquad (3.4)
\end{aligned}$$

At each inner simulation, the sample path at t is initialized as:

$$(\tilde{S}_{t,t}, \tilde{F}_{t,t}, \tilde{G}_{t,t}) = (S_t, F_t, G_t) \qquad (3.5)$$

Before withdrawal at t, the partial derivatives are initialized. $\tilde{F}_{t,t}$ is set to be proportional to $\tilde{S}_t$, such that:

$$\frac{\partial \widetilde{F}_{t,t}}{\partial S_t} = \frac{\partial F_t}{\partial S_t} = \frac{F_t}{S_t}, \qquad (3.6)$$

and $\tilde{G}_{t,t}$ and $\tilde{I}_{t,t}$ are fixed as constants, such that:

$$\frac{\partial \widetilde{G}_{t,t}}{\partial S_t} = \frac{\partial G_t}{\partial S_t} = 0, \quad \frac{\partial \widetilde{I}_{t,t}}{\partial S_t} = \gamma \frac{\partial I_t}{\partial S_t} = 0. \tag{3.7}$$

Note that in Equation 3.4, no calculation is needed for $F_t \leq I_t$. In this case, the sub-account is already depleted at time t and withdrawals are not allowed. Therefore, GMWB's future liabilities beyond time t are no longer affected by the stock price S_t, so no hedging is necessary. In other words, when $F_t < I_t$, the inner simulation can be skipped and set $\Delta_t = 0$.

3.2.3 Dynamic Hedging for Variable Annuities

Below we provide a scheme used to perform a multi-period nested simulation in estimating Profit and Loss (P&L) for one outer scenario.

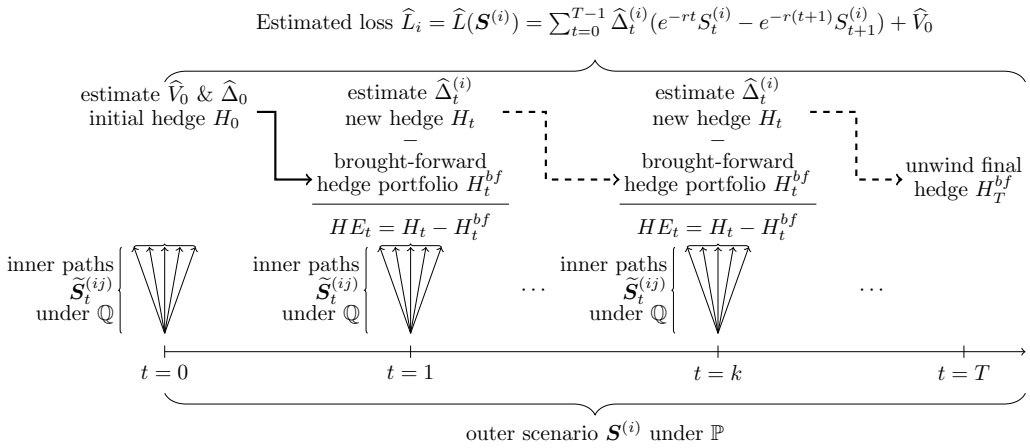


Figure 3.1: Illustration of a multi-period nested simulation that estimates the P&L for one outer scenario.

Insurers commonly use dynamic hedging to mitigate a market risk exposure in VA contract's embedded options. In a dynamic hedging program, a hedge portfolio is set up and periodically rebalanced for a portfolio of VA contracts using stocks, bonds, futures, and other derivatives. For simplicity, in this study we consider delta hedging for a generic VA liability using one stock and one bond. The metamodeling procedures presented in Section 3.3 and Section 3.4 can be trivially adapted to more general hedging strategies.

Consider hedging a VA contract whose sub-account is invested in a stock, and the hedging portfolio is rebalanced monthly. At $t \leq T$, a dynamic equation can be specified for the sub-account value F_t, in the absence of withdrawals, as:

$$F_t = F_{t-1} \cdot \frac{S_t}{S_{t-1}} \cdot (1 - \eta_g)$$

where $\eta_g > \eta_n$ is the gross rate of management fee that is deducted from the sub-account value at the end of each month. The delta hedge portfolio at any month t, $t = 0, 1, \ldots, T-1$, consists of Δ_t units in the underlying stock and B_t amount of a risk-free zero-coupon bond maturing at T months. The value of the hedge portfolio at month $(t-1)$ is:

$$H_{t-1} = \Delta_{t-1} S_{t-1} + B_{t-1},$$

where S_t is an underlying stock price and any month $t > 0$. Assuming there is no rebalacing between $t-1$ and t, this hedge portfolio is brought forward to month t,. Its value at month t becomes:

$$H_t^{bf} = \Delta_{t-1} S_t + B_{t-1} e^{-rt}.$$

Therefore, the time t hedging error, which is the cash flow incurred by the insurer due to rebalancing at time t, is

$$HE_t = H_t - H_t^{bf}, \quad t = 1, \ldots, T-1. \tag{3.8}$$

The P&L of the VA contract includes cost of the initial hedge (H_0), the periodic hedging gains and losses due to rebalancing at each time $t = 1, \ldots, T-1$ (Equation 3.8), unwinding of the hedge at maturity (H_T^{bf}), the payment of guaranteed benefit, and the management fee income. The present value of these cash flows, discounted at the risk-free rate, is the insurer's overall P&L from the VA contract. At $t = 0$, the insurer's overall P&L, i.e., the loss random variable L to which a risk measure is applied, is given by

$$\begin{aligned}
L &= H_0 + \sum_{t=1}^{T-1} e^{-rt} HE_t - e^{-rT} H_T^{bf} + V_0 \\
&= H_0 + \sum_{t=1}^{T-1} e^{-rt} (H_t - H_t^{bf}) - e^{-rT} H_T^{bf} + V_0 \\
&= B_0 + S_0 V_0 + \sum_{t=1}^{T-1} e^{-rt} \left(B_t + S_t \Delta_t - B_{t-1} \frac{e^{-r(t-1)}}{e^{-rt}} - S_{t-1} \Delta_{t-1} \right) - e^{-rT} S_T \Delta_{T-1} + V_0,
\end{aligned} \tag{3.9}$$

where V_0 can be thought of as a discounted payoff of the VA contract at $t = 0$, net of fee income in the absense of any hedging. The terms $H_0 + \sum_{t=1}^{T-1} e^{-rt} HE_t - e^{-rT} H_T^{bf}$ is the gain or loss from hedging. It consists of three components:

- H_0 is the initial cost of the hedging portfolio,
- $\sum_{t=1}^{T-1} e^{-rt} HE_t$ is the gain or loss from all rebalacing trades at time $t = 1, \ldots, T-1$, and
- $-e^{-rT} H_T^{bf}$ is the proceeds from unwinding the hedge at maturity T.

Rearranging the terms in Equation 3.9, the loss random variable L can be simplified as

$$L = \sum_{t=0}^{T-1} \Delta_t (e^{-rt} S_t - e^{-r(t+1)} S_{t+1}) + V_0. \tag{3.10}$$

Equation 3.10 shows that in bond holdings of hedging portfolio, all intermediate bond transactions cancel out since the interest rate r at which bond value accumulates is the same as rate at which P&Ls from bond transactions are discounted. In the stock holdings of the hedging portfolio, the gain or loss arises from the inital set-up of stock future holdings and the P&L from each rebalancing trade. In (3.10), Δ_t and V_0 are determined by using a risk-neutral measure $\mathbb{Q}$ while the distribution of L is under a real-world measure $\mathbb{P}$. If Δ_t and V_0 cannot be calculated analytically, a nested simulation is required to estimate the tail risk measure of L. Specifically, the outer scenarios $\boldsymbol{S}^{(i)} = (S_1^{(i)}, \ldots, S_T^{(i)})$, $i = 1, \ldots, M$ are generated under $\mathbb{P}$ measures to estimate the loss distribution. At each time $t = 1, \ldots, T-1$ of a given outer scenario $\boldsymbol{S}^{(i)}$, inner sample paths $\widetilde{\boldsymbol{S}}_t^{(j)} = (\widetilde{S}_{t+1}^{(j)}, \ldots, \widetilde{S}_T^{(j)})$, $j = 1, \ldots, N$ are generated under $\mathbb{Q}$ to estimate $\Delta_t^{(i)}$, $i = 1, \ldots, M$. Also, $V_0^{(i)}$, $i = 1, \ldots, M$ are estimated under $\mathbb{Q}$ via inner simulations at time 0. Recall from Section 3.2.2 that the stock's sample path, regardless of inner or outer simulation or a combination of both, determines the evolution of the triplet (S_t, F_t, G_t). For GMWB, a standard nested simulation procedure to estimate the α-CVaR of L is described in Algorithm 1 below.

We refer to the collection of experiments needed conditional on one scenario $\boldsymbol{S}^{(i)}$ to estimate L_i, which is all upward arrows in Figure 3.1, as one inner simulation. We make four observations:

- Each inner simulation is time-consuming, as it includes T simulation experiments: one at each time $t = 0, \ldots, T-1$,

Algorithm 1 Standard Nested Simulation Procedure for Estimating CVaR for GMWB Hedging Losses

1: For $i = 1, \ldots, M$, simulate outer scenarios $\boldsymbol{S}^{(i)} = (S_1^{(i)}, \ldots, S_T^{(i)})$ under the real-world measure $\mathbb{P}$.
2: For $t = 0$, simulate time-0 inner paths $\widetilde{\boldsymbol{S}}_0^{(j)} = (\widetilde{S}_1^{(j)}, \widetilde{S}_2^{(j)}, \ldots, \widetilde{S}_T^{(j)})$, $j = 1, \ldots, N$ under $\mathbb{Q}$, then estimate V_0 by $\widehat{V}_0 = \sum_{s=1}^{T} e^{-r(T-s)}[(I_s - F_s)^+ - \eta_n F_s]$ and $\widehat{\Delta}_0 = \Delta_0(\widetilde{S}_1^{(j)}, \ldots, \widetilde{S}_T^{(j)} | S_0)$.
3: Given each scenario $\boldsymbol{S}^{(i)}$:
   a. At each time $t = 1, \ldots, T-1$, simulate inner paths $\widetilde{\boldsymbol{S}}_t^{(ij)} = (\widetilde{S}_{t+1}^{(ij)}, \ldots, \widetilde{S}_T^{(ij)})$, $j = 1, \ldots, N$ under $\mathbb{Q}$, then estimate Δ_t by $\widehat{\Delta}_t^{(i)} = \Delta_t(\widetilde{S}_{t+1}^{(ij)}, \ldots, \widetilde{S}_T^{(ij)} | \boldsymbol{S}_t^{(i)})$.
   b. Use scenarios $\boldsymbol{S}^{(i)}$ and $\widehat{V}_0$ and $\widehat{\Delta}_t^{(i)}$ to calculate losses $\widehat{L}_i^{MC}$, $t = 0, \ldots, T-1$, then sort them as $\widehat{L}_{(1)}^{MC} \leq \widehat{L}_{(2)}^{MC} \leq \cdots \leq \widehat{L}_{(M)}^{MC}$.
4: Estimate α-CVaR of L by $\widehat{\text{CVaR}}_\alpha^{MC} = \frac{1}{(1-\alpha)M} \sum_{i=\alpha M+1}^{M} \widehat{L}_{(i)}^{MC} = \frac{1}{(1-\alpha)M} \sum_{i \in \widehat{\mathcal{T}}_{(1-\alpha)M}^{MC}} \widehat{L}_i^{MC}$.

- After running inner simulations for M scenarios, we obtain simulated data as feature-label pairs, $(\boldsymbol{S}^{(i)}, \widehat{L}_i)$, $i = 1, \ldots, M$; the feature vector $\boldsymbol{S}$ is T dimensional.
- $\widehat{L}_i^{MC}$ is a standard Monte Carlo estimator of the true loss for scenario $\boldsymbol{S}^{(i)}$. It is an unbiased estimator and its variance is inversely proportional to the number of inner replications N. As N approaches infinity, $\widehat{L}_i^{MC}$ converges to the true loss L_i, and
- Most importantly, when estimating tail risk measures such as α-CVaR, only a small number of estimated losses, which is those associated with the set of tail scenarios $\widehat{\mathcal{T}}_k$ are used in the estimator.

3.3 Two-Stage Nested Simulation with Metamodels

Based on the three observations above and built on the work by Dang et al. (2020), we propose a two-stage nested simulation procedure which uses a DNN metamodel to identify potential tail scenarios. We present our proposed procedure as a competitor to the standard procedure with M outer scenarios and N inner replications for each outer scenario, as described in Section 3.2.3. We propose a two-stage procedure with a supervised learning metamodel that aims at producing a CVaR estimate that is as accurate as that of the standard procedure, but requires fewer computations than the latter.

Algorithm 2 Two-Stage Metamodeling Nested Simulation Procedure for Estimating CVaR

1: **Train a neural network metamodel using simulation data:**

a. Use a fraction of the total simulation budget to run Steps 1, 2, and 3 in the standard procedure, i.e., Algorithm 1, with the same number of outer scenarios, M, but a much smaller number of inner replications, $N' \ll N$, in each scenario. Obtain M simulated samples (feature-label pairs), $(\boldsymbol{S}^{(i)}, \widehat{L}_i)$, $i = 1, \ldots, M$. Note that N' may be much smaller than N, so $\widehat{L}_i$ are expected to have larger variance.

b. Use the simulated data, $(\boldsymbol{S}^{(i)}, \widehat{L}_i)$, $i = 1, \ldots, M$ to train a supervised learning model. It is standard practice in supervised learning research to transform the stock prices $\boldsymbol{S}^{(i)}$ to returns $\mathbf{X}^{(i)}$,

$$\mathbf{X}^{(i)} = \left(\frac{S_1^{(i)} - S_0}{S_0}, \frac{S_2^{(i)} - S_1^{(i)}}{S_1^{(i)}}, \ldots, \frac{S_T^{(i)} - S_{T-1}^{(i)}}{S_{T-1}^{(i)}} \right) \tag{3.11}$$

and use $\mathbf{X}^{(i)}$ as features instead of the stock prices $\boldsymbol{S}^{(i)}$. Refer to the trained model as a metamodel and denote it by $\widehat{L}^{PD}(\mathbf{X})$. Denote the predicted losses for the outer scenarios by $\widehat{L}_i^{PD} = \widehat{L}^{PD}(\mathbf{X}^{(i)})$, $i = 1, \ldots, M$.

c. Sort the predicted losses $\widehat{L}_{(1)}^{PD} \leq \widehat{L}_{(2)}^{PD} \leq \cdots \leq \widehat{L}_{(M)}^{PD}$ to identify a predicted tail scenario set, $\widehat{\mathcal{T}}_m^{PD}$, associated with the largest predicted losses. The number of predicted tail scenarios, m, is a user's choice.

2: **Concentrate simulation on predicted tail scenarios:**

a. Run Steps 2 and 3 of Algorithm 1 with the same number of inner replications, N, but only on the predicted tail scenarios, i.e., scenarios in $\widehat{\mathcal{T}}_m^{PD}$. Denote the standard procedure's estimated losses and sorted losses by $\widehat{L}_i^{ML}$ and $\widehat{L}_{(i)}^{ML}$, respectively, $i = 1, \ldots, m$.

b. Estimate the α-CVaR of L by

$$\widehat{\mathrm{CVaR}}_\alpha^{ML} = \frac{1}{(1-\alpha)M} \sum_{i=\alpha M+1}^{M} \widehat{L}_{(i)}^{ML} = \frac{1}{(1-\alpha)M} \sum_{i \in \widehat{\mathcal{T}}_{(1-\alpha)M}^{ML}} \widehat{L}_i^{ML},$$

where $\widehat{\mathcal{T}}_k^{ML}$ denotes a predicted tail scenario set associated with the largest k estimated losses.

In the two-stage procedure, the neural network metamodel approximates L in Equation 3.10, where

$$L = L(\mathbf{S}_T) = L(\Delta_0, \ldots, \Delta_{T-1}, S_0, \ldots, S_T) \tag{3.12}$$

The loss L depends on all the hedging weights $\Delta_0, \ldots, \Delta_{T-1}$, which are determined by the stock prices S_t, $t = 0, \ldots, T-1$. In addition, L also depends on all the stock prices $S_0, \ldots, S_T$ directly. Therefore, the loss $L(\mathbf{S}_T)$ is a complex function of the entire stock price path $\mathbf{S}_T$, and the neural network metamodel $\widehat{L}^{PD}(\mathbf{X})$ approximates $L(\mathbf{S}_T)$ by mapping the stock returns $\mathbf{X}_T$ to the loss $L(\mathbf{S}_T)$. Statistically speaking, $L(\mathbf{S}_T)$ is a random variable whose stochasticity solely depends on the stock price path $\mathbf{S}_T$. Nevertheless, $L(\mathbf{S}_T)$ can be viewed as a function of the underlying stock returns $\mathbf{X}_T$ and can be approximated by a neural network metamodel.

Similar to Dang et al. (2020), the proposed two-stage procedure in Algorithm 2 uses the metamodel predictions to identify the predicted tail scenario set in stage 1. However, different from their fixed-budget simulation design, we attempt to achieve a target accuracy. Specifically, in stage 2 we propose using a standard procedure with the same number of inner replications, N, as a benchmark procedure. There are two different types of experiment designs for nested simulation procedures: fixed-budget design and fixed-accuracy design. In a fixed-budget design, the simulation budget is fixed, and the goal is to achieve the highest degree of accuracy possible within the budget. Let $\Gamma = MN$ be the simulation budget for the standard procedure, where each scenario receives $\frac{\Gamma}{M}$ inner replications. In the proposed two-stage procedure, suppose that stage 1 uses 1% of the simulation budget, $\alpha = 95\%$, and $m = (1-\alpha)M$, then 99% of the simulation budget is concentrated on $5\%M$ predicted tail scenarios in stage 2. In other words, each predicted tail scenario receives $\frac{99\%\Gamma}{5\%M}$ inner replications, almost 20 times more than that in the standard procedure. This budget concentration is expected to improve the estimation accuracy of the two-stage procedure compared to a standard procedure with the same budget. If the metamodel is accurate in predicting true tail scenarios, the two-stage procedure is expected to achieve a higher degree of accuracy than the standard procedure with the same budget. However, we believe that the goal of designing an efficient simulation procedure is to solve practical problems faster, so a target-accuracy design is more suitable, which refers to obtaining a similar level of accuracy as the standard procedure but with much less simulation budget. One other reason for recommending this fixed-accuracy design is to investigate whether DNN metamodels trained with much noisier labels can identify true tail scenarios with a similar degree of accuracy as the standard procedure. The size of the predicted tail scenario set in stage 1, m, is an important experiment design parameter that affects the correct identification of true tail scenarios and ultimately affects the estimation accuracy for CVaR. Clearly, there is a lower bound $m \geq (1-\alpha)M$ because the α-CVaR is estimated

by the average of the largest $(1-\alpha)M$ losses at the end of stage 2. For ease of reference, we call the additional percentage of predicted tail scenarios above this lower bound, i.e., $\epsilon = \frac{m-(1-\alpha)M}{M}$, as a *safety margin*. On one hand, large ϵ is not desirable because it increases computations in stage 2. On the other hand, ϵ should be set reasonably large so that more true tail scenarios are included in the predicted tail scenario set $\widehat{\mathcal{T}}_m^{PD}$ and are ultimately included in $\widehat{\mathcal{T}}_{(1-\alpha)M}$ at the end of stage 2. The selection of m is highly dependent on the choice of the metamodel. Due to the simulation errors and approximation error in the metamodel in stage 1, we do not expect a perfect match between the true tail scenario set $\mathcal{T}_k$ and the predicted tail scenario set $\widehat{\mathcal{T}}_k^{PD}$ for any size k. This means that we should not set m at its lower bound: Some safety margin ϵM should be added to the predicted tail scenario set, i.e., $m = (1-\alpha)M + \epsilon M$, to increase the likelihood that $\mathcal{T}_k \subseteq \widehat{\mathcal{T}}_m^{PD}$ and that the true tail scenarios are included in estimating α-CVaR at the end of stage 2. The choice of a safety margin ϵ is not trivial, and it should be set based on the metamodel's accuracy in identifying true tail scenarios. In the numerical experiments, we examine the relationship between the safety margin and the correct identification of true tail scenarios for different metamodels.

3.4 Single-Stage Nested Simulation with Neural Network Metamodels

In our exploratory experiment with the two-stage procedure, we observe that a suitable metamodel trained with noisy labels is accurate enough to identify true tail scenarios with a relatively small safety margin. This observation motivates us to propose a single-stage procedure that uses the identical neural network metamodel, not for the purpose of differentiating between tail and non-tail scenarios, but rater to estimate the CVaR directly using the predicted losses.

The single-stage procedure has three major advantages over the two-stage procedure:

1. The single-stage procedure is expected to be more efficient than the two-stage procedure because it does not require us to run the standard procedure described in stage 2.

2. The safety margin ϵ is not needed in the single-stage procedure. The safety margin is difficult to determine because it depends on the metamodel's accuracy in identifying true tail scenarios. The single-stage procedure does not need a safety margin because it directly estimates the risk measure by using the predicted losses.

Algorithm 3 Single-Stage Metamodeling Nested Simulation Procedure for Estimating CVaR

1: Run Step 1 of Algorithm 2, and denote the predicted losses by $\widehat{L}_i^{PD} = \widehat{L}^{PD}(\boldsymbol{S}^{(i)})$, $i = 1, \ldots, M$.
2: Sort the predicted losses $\widehat{L}_{(1)}^{PD} \leq \widehat{L}_{(2)}^{PD} \leq \cdots \leq \widehat{L}_{(M)}^{PD}$ to identify the largest predicted losses.
3: Directly estimate the risk measure (e.g., α-CVaR) of L using the metamodel predictions: Calculate $\widehat{\text{CVaR}}_\alpha^{PD} = \frac{1}{(1-\alpha)M} \sum_{i=\alpha M+1}^{M} \widehat{L}_{(i)}^{PD}$.

3. Most importantly, the single-stage procedure is not limited to just estimating tail risk measures and can be extended to provide a broader assessment of risk. It can be naturally adapted to estimate risk measures that require the knowledge of the entire loss distribution, such as the standard deviation or the squared tracking error. Even for CVaR, the two-stage procedure becomes less attractive at a higher level of α.

3.5 Numerical Results

We conduct a series of simulation experiments to

1. demonstrate the efficiency of the proposed metamodeling procedures, and
2. examine the error tolerance to noisy training data in deep learning models.

The problem settings in our experiments are identical to those in Dang et al. (2020). We consider estimating the 95% CVaR of the hedging loss of a GMWB contract, which is one of the most complex VA contracts in the market. The VA contracts have a 20-year maturity and are delta-hedged with monthly rebalancing, i.e., $T = 240$ rebalancing periods. The gross and net management fees are $\eta_g = 0.2\%$ and $\eta_n = 0.1\%$, respectively. The withdrawal rate for GMWB is 0.375% per month. To make our examples more realistic, the VAs are also subject to a dynamic lapse model with the following key features (NAIC, 2021):

1. When a policyholder lapses, both the fund value (F) and guarantee value (G) are reduced proportionally.

2. The monthly lapse rate from time t to $t+1$, denoted as ${}_{\frac{1}{12}}q^{l}_{x+\frac{t}{12}}$, is given by:

$$ {}_{\frac{1}{12}}q^{l}_{x+\frac{t}{12}} = \min\left\{1, \max\left\{0.5, 1 - 1.25 \times \left(\frac{G_t}{F_t} - 1.1\right)\right\} \times {}_{\frac{1}{12}}q^{base}_{x+\frac{t}{12}}\right\} \tag{3.13} $$

where ${}_{\frac{1}{12}}q^{base}_{x+\frac{t}{12}}$ is a base monthly mortality rate for a policyholder aged $x + \frac{t}{12}$ at month t, and the base monthly lapse rate is defined as:

$$ {}_{\frac{1}{12}}q^{base}_{x+\frac{t}{12}} = \begin{cases} 0.00417 & \text{if } t < 84, \\ 0.00833 & \text{if } t \geq 84. \end{cases} \tag{3.14} $$

The risk-free rate is 0.2% per month and the underlying asset S_t is modeled by a regime-switching geometric Brownian motion with parameters specified in Table 3.1.

Parameter	**Monthly Value**
Risk-free rate	0.002
Mean return for regime 1	0.0085
Mean return for regime 2	−0.0200
Standard deviation for regime 1	0.035
Standard deviation for regime 2	0.080
Transition probability from regime 1	0.04
Transition probability from regime 2	0.20

Table 3.1: Real-world parameters for the regime-switching model (monthly rates)

To compare the numerical performances of different simulation procedures, we create a benchmark dataset with a large-scale nested simulation: We first simulate $M = 100{,}000$ outer scenarios, i.e., 240-periods stock paths $\boldsymbol{S}^{(1)}, \ldots, \boldsymbol{S}^{(M)}$ under $\mathbb{P}$ and used these outer scenarios in all further experiments. Note that the 5% tail scenario set includes 5000 scenarios. As the hedging losses for these scenarios cannot be calculated analytically, we run inner simulations with a large number of replications, $N = 100{,}000$, conditional on each of the M scenarios. We denote these losses by $L_1, \ldots, L_M$ and will refer to them as *true losses*. We also use these true losses to estimate $\widehat{\text{CVaR}}_{95\%}$ and denote the corresponding *true tail scenario set* by $\mathcal{T}_{5000}$. Lastly, we refer to the set of feature-label pairs $\{(\boldsymbol{S}^{(i)}, L_i) : i = 1, \ldots, M\}$ as a *true dataset*. Note that the feature vector $\boldsymbol{S}$ is a 240-dimension stock path.

In our experiments, the standard nested simulation procedure is mainly used in three ways.

1. The true losses and the true 95%-CVaR are estimated by using a standard nested simulation procedure that runs $N = 100{,}000$ inner replications for each of the $M = 100{,}000$ outer scenarios.

2. A benchmark estimator of the 95%-CVaR is generated by using $N = 1000$ for each of the $M = 100{,}000$ outer scenarios.

3. The training data for our metamodels is generated by using $N = 100$ for $M = 100{,}000$. The value of M and N are subject to change in our other numerical experiments.

Five metamodels are considered in this experiment: Multiple Linear Regression (MLR), Quadratic Programming Regression (QPR) without interaction terms, FNN, RNN, and LSTM network. MLR and QPR are two common metamodels in the simulation literature, and their implemention is straightfoward in our simulation setting. FNN is a generic neural network while RNN and LSTM are specialized models to accommodate the sequential structure of our time-series features. A tanh activation function is used for RNN and LSTM layers, and a ReLU activation function is used for the fully-connected layers. All neural network metamodels are trained by the Adam optimizer (Kingma and Ba, 2014) with an initial learning rate of 0.001 and an exponential learning rate decay schedule. All DNN metamodels are trained with a dropout rate of 10%. The architectures and training settings are typical choices in the deep learning literature. The training labels are normalized to have a zero mean and a unit standard deviation. The numbers of trainable parameters are recorded in the second column of Table 3.2. We observe that the three DNN metamodels have orders of magnitudes more trainable parameters than the two traditional regression metamodels. Following the convention in machine learning literature, the three DNN metamodels are said to have much higher *model capacities*. Treating regression metamodels as shallow neural networks, MLR and QPR are two neural networks with no hidden layer. For a traditional regression metamodel, the numbers of neurons in its input layer equals to the cardinality of its regression basis. On the other hand, the three DNN metamodels have several hidden layers and are more flexible than MLR and QPR. Our FNN has two hidden layers prior to its output layer. Its hidden layers are feature extraction layers that transform the 240-dimension feature vector into a 32-dimension vector that can directly relate to the target variable. This extraction process is automatic and does not require manual feature engineering. Our RNN and LSTM networks have three hidden layers. They contain similar numbers of neurons as the FNN but have deeper network structures. For RNN, two of its hidden layers are recurrent layers that capture the temporal dependency in the feature, and its last hidden layer is a standard

feedforward layer that transforms the previous hidden states into a 32-dimension vector for target prediction. Our LSTM has the same architecture as the RNN but with LSTM cells in place of RNN cells.

Metamodel	Capacity	Training error	Test error	True error
MLR	241	$0.706(\pm 8.34 \times 10^{-4})$	$0.713(\pm 2.67 \times 10^{-2})$	$0.706(\pm 3.44 \times 10^{-4})$
QPR	481	$0.543(\pm 8.27 \times 10^{-4})$	$0.554(\pm 2.32 \times 10^{-2})$	$0.544(\pm 4.12 \times 10^{-4})$
FNN	35,009	$0.129(\pm 5.95 \times 10^{-3})$	$0.240(\pm 9.82 \times 10^{-3})$	$0.132(\pm 5.82 \times 10^{-3})$
RNN	32,021	$0.132(\pm 7.53 \times 10^{-3})$	$0.137(\pm 7.62 \times 10^{-3})$	$0.119(\pm 7.51 \times 10^{-3})$
LSTM	35,729	$0.075(\pm 4.48 \times 10^{-3})$	$0.079(\pm 5.35 \times 10^{-3})$	$0.063(\pm 4.43 \times 10^{-3})$
RNN*[1]	32,021	$0.109(\pm 5.20 \times 10^{-3})$	$0.128(\pm 5.22 \times 10^{-3})$	$0.109(\pm 5.20 \times 10^{-3})$

Table 3.2: Architectures and MSEs of metamodels for GMWB inner simulation model.

For each metamodel, 100 independent macro replications are run. Each macro replication involves simulating a noisy dataset, randomly separating it into training and test data, fitting a metamodel, and forming predictions. The last three columns in Table 3.2 display the average squared errors between the metamodel predictions and the loss labels for different datasets. Training error, test error, and true error are the average squared errors between the metamodel predictions and the training labels, test labels, and true labels, respectively. The half lengths of their 95% confidence bands are also reported in parentheses. We first observe that the two regression metamodels have high training errors. This is a sign of under-fitting. Model capacities of the regression metamodels are too low to learn the training dataset. A successful learning on the training dataset is a necessary condition for a metamodel to generalize to the true simulation model. All neural network metamodels have high model capcities comparing with the number of data points in the training set, i.e., 90,000. Empirically, this is not a huge concern for the DNNs. In neural network literature, the number of trainable parameters is often much larger than the number of training samples. AlexNet (Krizhevsky et al., 2012), a well-known CNN, has 60 million trained parameters and is trained on the ImageNet dataset with only 1.2 million sample images. Other examples include generative pre-trained transformer models like BERT (Devlin, 2018) and GPT-3 (Brown, 2020), which have billions of trainable parameters and are trained on large-scale text corpora. The reason for not over-fitting when the number of trainable parameters is much larger than the number of training samples is still an open question in deep learning community. One of the explanations in the literature is that the optimization surface of DNNs is highly non-convex and has many local minima.

[1]This row summarizes the error metrics of the well-trained RNN, where the ones suffer from a vanishing gradient issue are removed from the averages.

Given a good initialization, the optimization algorithm is likely to find a solution that generalizes well to unseen data. Another explanation is that most existing DNN models have specific architectures that make them less prone to over-fitting. For example, a CNN is specifically designed to capture a spatial structure in image data. Unlike the FNN, not all neurons are connected to all neurons in the previous layer. This architecture serves as prior knowledge and a form of regularization that make CNNs less prone to over-fitting.

In our experiments, there are important differences among the three DNN models due to having different neural network architectures. Their model capacities are intentionally set to be similar to each other for a fair comparison. The FNN is a generic neural network, its test error is much greater than the training error, which is a sign of over-fitting and poor generalization. In contrast, RNN and LSTM networks have recurrent structures and are designed to capture temporal relationship in the high-dimensional feature. As a result, they have lower training errors. They fit the data better and have lower test errors, i.e., they generalize better. Notably, the true errors for the DNN metamodels are lower than the training errors. This is a sign of the metamodels' ability to learn the true feature-label relationship with noisy training labels. Section 3.5.2 investigates this phenomenon in greater details.

The intuition behind the RNN and LSTM metamodels is briefly mentioned in 1.3.5. To overcome the overfitting issue of the FNN, RNN learns a single function that operates across all time steps, and the time t itself is not a variable in the RNN. The RNN cells in RNN and LSTM networks share parameters across time and serve as a form of regularization that prevents overfitting. However, the price that RNN pays for a reduced number of parameters is the difficulty in its optimization. Comparing the two metamodels with recurrent structures, a LSTM network overcomes two key drawbacks of a crude RNN.

1. It avoids the vanishing gradient problem during training, which makes the LSTM network easier to train than the RNN. Figure 3.2 shows the quantile-quantile (QQ) plots between the **training** labels and the RNN predictions for two macro replications, where training of the latter is hindered by the vanishing gradient problem. As a result, the half-width of the 95% confidence band of the RNN's training error is much larger than all other metamodels. The last row of Table 3.2 shows the average squared errors of the good RNN metamodels that have been successfully trained. Removing the RNN metamodels that are ill-trained, the average squared errors of the RNN metamodels are lower than those of the FNN. This suggests that the RNN is more sensitive to noise in the sense of a stochastic gradient descent than the FNN. They can capture the temporal dependency but are more difficult to optimize.

2. A LSTM introduces three gates that regulate the flow of information. These gates

decide what information should be kept or discarded. For detailed discussion on the LSTM network, please refer to Section 1.3.5. This gating mechanism makes a LSTM network more capable of learning long-term dependencies. Since the feature in our dynamic hedging problem is a 240-dimensional time series, a LSTM network is more suitable than a crude RNN.

In summary, the network depth ensures successful learning of the training dataset, and the network architecture serves as The intuition behind the RNN and LSTM metamodels is briefly mentioned in 1.3.5. To overcome the overfitting issue of the FNN, RNN learns a single function that operates across all time steps, and the time t itself is not a variable in the RNN. The RNN cells in RNN and LSTM networks share parameters across time and serve as a form of regularization that prevents overfitting. However, the price that RNN pays for a reduced number of parameters is the difficulty in its optimization. Comparing the two metamodels with recurrent structures, a LSTM network overcomes two key drawbacks of a crude RNN.

1. It avoids the vanishing gradient problem during training, which makes the LSTM network easier to train than the RNN. Figure 3.2 shows the quantile-quantile (QQ) plots between the **training** labels and the RNN predictions for two macro replications, where training of the latter is hindered by the vanishing gradient problem. As a result, the half-width of the 95% confidence band of the RNN's training error is much larger than all other metamodels. The last row of Table 3.2 shows the average squared errors of the good RNN metamodels that have been successfully trained. Removing the RNN metamodels that are ill-trained, the average squared errors of the RNN metamodels are lower than those of the FNN. This suggests that the RNN is more sensitive to noise in the sense of a stochastic gradient descent than the FNN. They can capture the temporal dependency but are more difficult to optimize.

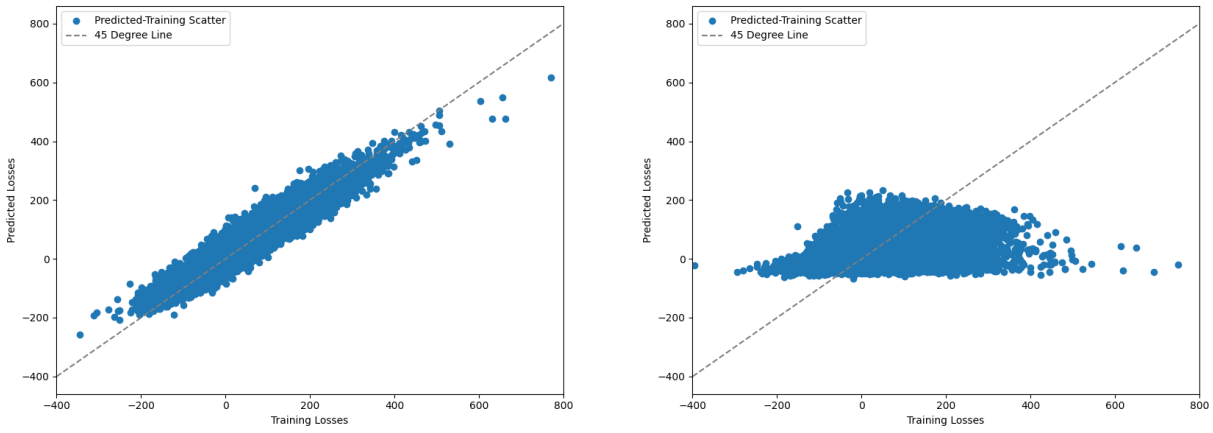
2. A LSTM introduces three gates that regulate the flow of information. These gates decide what information should be kept or discarded. For detailed discussion on the LSTM network, please refer to Section 1.3.5. This gating mechanism makes a LSTM network more capable of learning long-term dependencies. Since the feature in our dynamic hedging problem is a 240-dimensional time series, a LSTM network is more suitable than a crude RNN.

In summary, the network depth ensures successful learning of the training dataset, and the network architecture serves as

1. regularization that prevents over-fitting, and

2. prior knowledge that guides the optimization algorithm to find a solution that generalizes well to unseen data.

They are both crucial for the metamodels' performance. It is important to choose a *suitable* architecture with a pre-set capacity that is large enough to learn the training dataset and generalize to the true feature-label relationship.

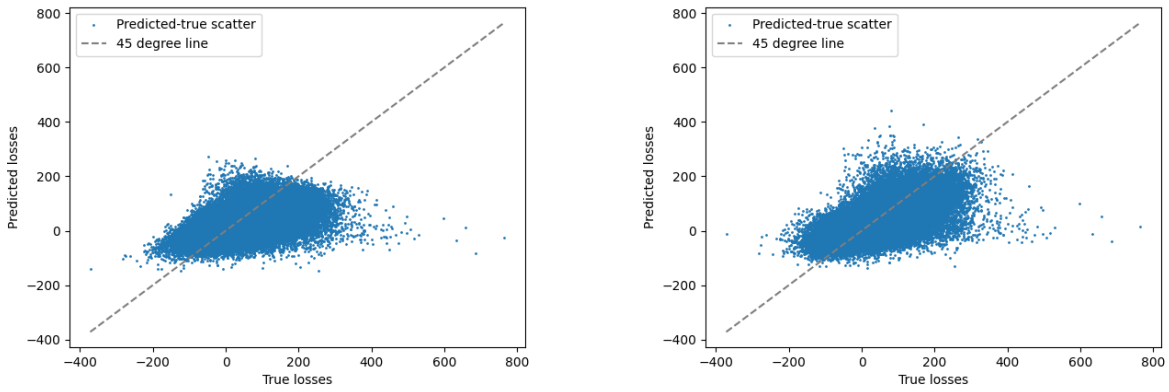


(a) A good RNN metamodel

(b) A bad RNN metamodel

Figure 3.2: QQ-plots between true labels (x-axis) and predicted losses (y-axis) for the RNN metamodel.

Recall that all three datasets are generated from the same simulation model, but the true labels contain much less simulation noise than the training and test labels. Part of the training error is from the simulation noise, and this noise is much less reflected in the true error. We observe that the two regression metamodels is under-fitting with high training errors. They also have low test and true errors, which indicates poor generalization. In contrast, the DNN metamodels have lower errors. More specifically, they generalize better to the true labels than to the test labels. The test labels contain more simulation noise than the true labels. This is a sign of the regression metamodels' inability to learn the true feature-label relationship rather than merely over-fitting to the simulation noise in the training labels. In practical applications, the true labels are not available. However, analyzing the quality of fit on the true labels in a simulation environment offers a unique opportunity to understand the metamodels' true ability of learning. Figure 3.3 and Figure 3.4 shows the QQ plots between the **true** loss labels and the predicted losses for the regression metamodels and the neural network metamodels, respectively.



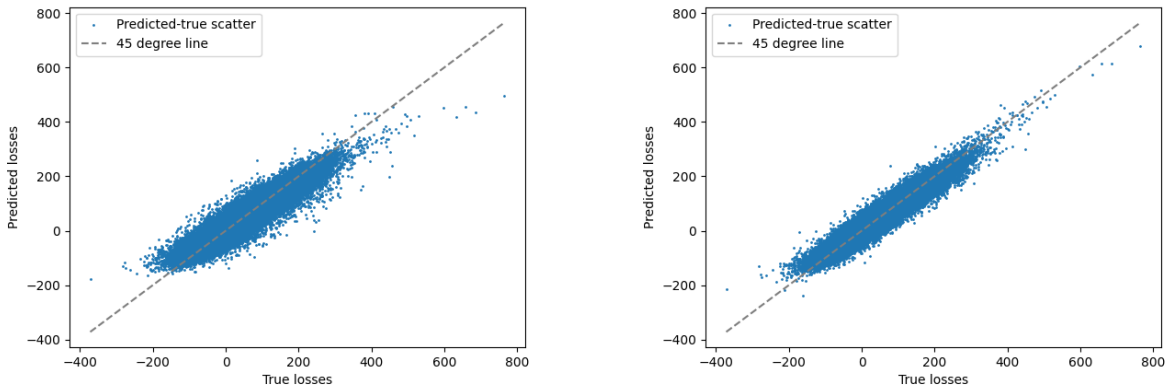
(a) MLR metamodel

(b) QPR metamodel

Figure 3.3: QQ-plots between true losses (x-axis) and predicted losses (y-axis) for regression metamodels.

Comparing with the MSE table in Table 3.2 that summarizes the overall fit, QQ plots offer a closer look at the metamodels' performance on different parts of the loss distribution. Figure 3.3 show the regression metamodel predictions and the loss labels in the true dataset. Between the two regression metamodels, the QPR metamodel has a slightly better fit for larger losses. Nevertheless, the predictions of both regression metamodels are far from the true labels, and their fit on the tail is particularly undesirable. The poor fit on the tail hinders the regression metamodels' ability to identify true tail scenarios and ultimately leads to poor CVaR estimates. Adding the quadratic terms, our QPR metamodel is considered as a natural extension of the MLR. Attempts to further improve the regression metamodels by adding interaction terms or higher-order terms do not improve the fit. Feature engineering (i.e., variable selection) for regression metamodels is not a trivial task and is highly dependent on the simulation procedure. Having 240 time-series features further complicates this issue. As a result, our attempts to improve the traditional regression metamodels are not successful. The regression metamodels are not flexible enough to capture the complex feature-label relationship in the dynamic hedging simulation model.

In Figure 3.4, we illustrate the fit of the neural network metamodels. The FNN has a better fit than the regression metamodels, but it still has a poor fit on the tails. We observe that the LSTM metamodels' QQ-plots closely follow the 45-degree line. Again, these metamodels are trained on noisy data, so the good fit to the true data should not be taken for granted. This implies that these models indeed learn the true feature-label



(a) FNN metamodel

(b) LSTM metamodel

Figure 3.4: QQ-plots between true losses (x-axis) and predicted losses (y-axis) for neural network metamodels.

relationship in the dynamic hedging simulation model (i.e., true loss labels) even though they are trained on noisy observations (e.g., training labels) of the model.

From a unique analytical standpoint, our numerical study offers a methodical exploration into the robustness of DNN models against noise in training labels. By employing the standard nested simulation procedure in Algorithm 1 as a data generator, we gain the ability to manipulate the noise level by adjusting the number of inner replications while keeping the same simulation procedure. This approach provides a controlled environment to examine the impact of label noise on neural network models. It allows us to generate our true dataset that approximates the true hedging losses with a high degree of precision, and, as a result, it enables us to explore the crucial question of whether deep learning models are capable of learning the true feature-label relationship from noisy training labels. Our numerical results in Table 3.2 and the QQ plots provide direct evidence that DNN models are indeed able to cut through the noise in the training labels and learn the true feature-label relationship. The LSTM metamodels' ability to learn the true feature-label relationship is crucial for the two-stage procedure to identify true tail scenarios and produce accurate CVaR estimates. Section 3.5.2 discusses LSTM metamodels in more details regarding their sensitivity to data quality and quantity. We aim at providing numerical evidence and insights into the DNN metamodels' noise tolerance.

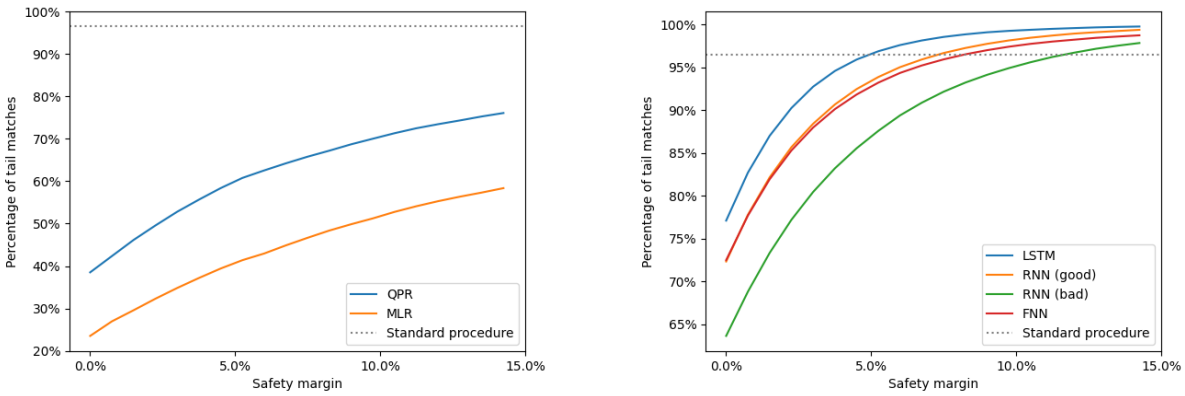
3.5.1 Two-Stage Procedure

We compare our two-stage procedure to a benchmark standard nested simulation procedure that runs $N = 1000$ inner replications for each of the $M = 100{,}000$ outer scenarios. In stage 1 of the proposed procedure, we first run inner simulations with $N' = 100$ inner replications for each of the $M = 100{,}000$ outer scenarios. So, stage 1's simulation budget is 10% of the standard procedure's. The resulting feature-label pairs $\{(\boldsymbol{S}^{(i)}, \widehat{L}_i) : i = 1, \ldots, M\}$ is used for training different metamodels. Specifically, following the convention in machine learning literature, we split this dataset into three parts: The training, validation, and test sets have 90,000, 5000, and 5000 samples (representing 90%, 5%, and 5% of the dataset), respectively. At the end of stage 1, m predicted tail scenarios are identified by the trained metamodels. In stage 2, $N = 1000$ inner replications are run for all predicted tail scenarios. This is the same number of replications as the benchmark standard procedure. Stage 2's simulation budget is $\frac{m}{M}$ of the benchmark standard procedure's. In short, the two-stage procedure uses $15\%-30\%$ of the benchmark procedure's budget for a safety margin between $0\% - 15\%$. Analyzing the two stages separately, the simulation budget for stage 1 is 10% of the benchmark procedure's. Stage 2 uses 5% to 20% of the simulation budget of the benchmark procedure for a safety margin between $0\% - 15\%$.

In our proposed two-stage procedure, the metamodel is used to identify the predicted tail scenario set, on which the standard nested simulation procedure is run in stage 2 to estimate the 95%-CVaR. An accurate identification of the true tail scenarios is crucial. In a two-stage procedure, metamodels' overall prediction errors measured by the MSEs in Table 3.2 are not the determining factor, but their ability to effectively rank the scenarios by their **true** losses is most critical in producing accurate CVaR estimates. A metamodel that uses less safety margin to rank can save more simulation budget in stage 2.

Figure 3.5 depicts the average percentage of correctly identified true tail scenarios by different metamodels for increasing safety margins. Each quantity is averaged over 100 macro replications. For a fixed safety margin, the percentage of tail matches of the traditional regression metamodels are substantially lower than the ones of the neural networks. We observe that a poor metamodel such as the QPR identifies less than 40% of the true tail scenario without any safety margin. In contrast, a well-suited metamodel such as the LSTM identifies more than 75% of the true tail scenarios without any safety margin and more than the QPR metamodel does with 15% of safety margin[2]. The RNN metamodel

[2]The metamodel in Dang et al. (2020) identifies 100% of the true tail scenarios on with a 10% safety margin for a GMAB contract. The GMAB contract is simpler than the GMWB contract, and the true tail scenarios are easier to identify. For a GMAB, our LSTM metamodel identifies 100% of the true tail scenarios with a 2.5% safety margin



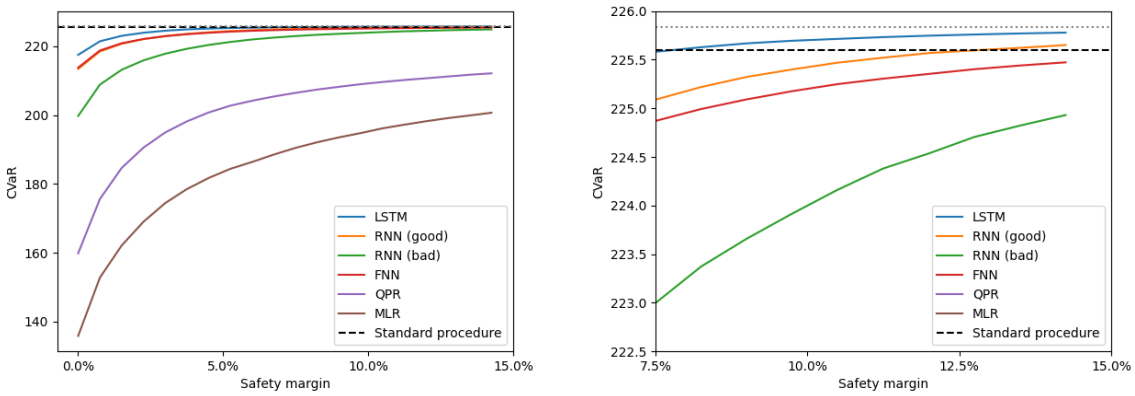
(a) Regression metamodels

(b) Neural network metamodel

Figure 3.5: Percentage of correctly identified true tail scenarios by different metamodels.

suffers from the vanishing gradient problem during the training stage. For some macro replications (as shown in Figure 3.2b), the RNN metamodels' predictions are far from the true losses, especially on the tails. Seperating the good training macro replications from the bad ones, the results reconcile with the findings in Table 3.2. The FNN metamodel has a better fit than the regression metamodels, but it does not have the same level of accuracy as the LSTM metamodel, which is a specialized network to capture the sequential structure of our time-series features. For comparison, the horizontal dotted line shows the percentage of correctly identified true tails for the standard nested simulation procedure. A good metamodel should be able to identify a similar percentage of true tail scenarios as the standard procedure does with a reasonable safety margin. Otherwise, the two-stage procedure will offer no computational advantage in simulation budget over the standard procedure. The LSTM metamodel can reach similar percentage of correctly identified true tail scenarios with a 5% safety margin. This indicates that the LSTM metamodel should be able to reach the same level of accuracy as the standard procedure with only 20% of the simulation budget.

Lastly, we return to our original goal of estimating the 95%-CVaR of the dynamic hedging error. Figure 3.6 shows the 95%-CVaR estimates for all five metamodels with different safety margins, averaged over 100 macro replications. Figure 3.6b is a zoomed version of Figure 3.6a. Because the safety margin only affects the two-stage procedure, the true 95%-CVaR and the one estimated by the standard procedure are horizontal lines (which are the solid and dotted lines, respectively) in Figure 3.6. We observe that, with reasonable safety



(a) Safety margin 0% - 15%.

(b) Safety margin 7.5% - 15%.

Figure 3.6: Average 95%-CVaR estimates by different procedures. The right figure is a zoomed-in version of the left figure.

margins, the two-stage procedures with a LSTM metamodel consistently produce estimates that are as accurate as the standard procedure's estimate. The amount of computational savings is substantial. The LSTM is particularly superior to other metamodels as it accurately identifies the true tail scenarios and produces highly accurate CVaR estimates with small safety margins. By concentrating the simulation budget on the predicted tail scenarios, the two-stage procedure with the LSTM metamodel is able to achieve a similar degree of accuracy as the standard procedure with a much smaller simulation budget. As the percentage of correctly identified tails approaches 100%, the two-stage procedure's CVaR estimates does not converge to the true CVaR but to the standard procedure's estimate. This is because the two-stage procedure's CVaR estimates are based on the standard procedure's estimates on the predicted tail scenarios, and the standard procedure's estimates themselves are also noisy.

3.5.2 Noise Tolerance of DNN Metamodels

For financial and actuarial applications, regulators and practitioners are often concerned about the robustness of DNN models to noise in training labels, which hinders the adoption of these models in practice. Since the true relationship is unknown in real-world applications, most deep learning literature illustrates the impact of noise by artificially injecting noise into real-world datasets, which is already noisy prior to the injection. In

our numerical experiments, we are able to use Monte Carlo simulation to generate a true dataset that approximates the true hedging losses with a high degree of precision, and, as a result, we are able to explore the important question of whether deep learning models are indeed capable of learning the true feature-label relationship from noisy training labels. In this section, we treat the standard nested simulation procedure as a data generator and examine the noise tolerance of LSTM metamodels by varying the numbers of the outer scenarios (M) and the inner replications (N) used to generate the training samples.

The number of outer scenarios corresponds to the number of feature-label pairs in the training dataset, and the number of inner replications controls the noise level in the training labels. Recall that we use the standard nested simulation procedure with $N = 100$ inner replications in our previous experiments, and we will refer to the resulting training dataset as the *low-noise dataset*. We also generate a *medium-noise dataset* and a *high-noise dataset* by running the standard nested simulation procedure with $N = 10$ and $N = 1$ inner replications, respectively. By altering the data quantity and quality, we conduct a sensitivity analysis on the LSTM metamodels' noise tolerance. We study the impact of noisy data on two LSTM metamodels with different model capacities, i.e., different numbers of trainable parameters. The two LSTM metamodels has the same number of layers, but their numbers of hidden units in each layer are different. More specfically, the high-capacity LSTM metamodel has 128 and 16 hidden units in the first and second LSTM layers, respectively, while the low-capacity LSTM metamodel has 32 and 4.

The maximum number of training samples is set to $M = 100{,}000$. The training samples are split into training and test sets with a 90%-10% split ratio. The true labels are generated by running the standard nested simulation procedure with $N = 100{,}000$ inner replications for each of the 100,000 outer scenarios. The reason for using a maximum of 100,000 data points is precisely due to the computational cost of running the standard nested simulation procedure. For consistency, the LSTM metamodels are trained with the same architecture and training settings as in the previous experiments.

The first two rows of Table 3.3 shows the average squared errors and the half widths of their 95% confidence intervals between the metamodel predictions and the labels in the training dataset and the true dataset. N indicates the noise level in the training labels, i.e., a simulation dataset with fewer inner replications has a higher noise level. The test errors are included as a practical measure of the metamodels' generalization ability to unseen noisy samples. For this particular experiment, we are more interested in the true errors, which measure the metamodels' ability to learn the true feature-label relationship from the low-noise, medium-noise, and high-noise datasets. We first observe that all the true errors are lower than the training errors, indicating that the LSTM metamodels generalize well to predicting the true losses. Both LSTM metamodels are able to learn the true feature-label

Model	N	Training error	Test error	True error
LSTM	100	$0.075(\pm 4.5 \times 10^{-3})$	$0.079(\pm 5.4 \times 10^{-3})$	$0.063(\pm 4.4 \times 10^{-3})$
High-capacity LSTM	100	$0.068(\pm 3.6 \times 10^{-3})$	$0.102(\pm 6.1 \times 10^{-2})$	$0.060(\pm 3.6 \times 10^{-3})$
Average Difference	100	-0.007	0.023	-0.003
LSTM	10	$0.195(\pm 1.1 \times 10^{-3})$	$0.193(\pm 1.7 \times 10^{-3})$	$0.070(\pm 9.1 \times 10^{-4})$
High-capacity LSTM	10	$0.157(\pm 2.0 \times 10^{-3})$	$0.199(\pm 1.9 \times 10^{-3})$	$0.065(\pm 1.9 \times 10^{-3})$
Average Difference	10	-0.038	0.006	-0.005
LSTM	1	$1.366(\pm 8.6 \times 10^{-3})$	$0.781(\pm 6.0 \times 10^{-3})$	$0.129(\pm 5.0 \times 10^{-3})$
High-capacity LSTM	1	$1.354(\pm 2.7 \times 10^{-2})$	$0.795(\pm 8.3 \times 10^{-2})$	$0.149(\pm 2.7 \times 10^{-2})$
Average Difference	1	-0.012	0.014	0.020

Table 3.3: MSEs of LSTM metamodels.

relationship from the low-noise and medium-noise datasets. However, when the noise level is high, both LSTM metamodels have high true errors.

The differences in the errors between the LSTM metamodels are also reported in the last rows of Table 3.3. A positive difference indicates that the high-capacity LSTM has a higher error than the regular LSTM. We observe that both LSTM metamodels have similar errors on the low-noise and medium-noise datasets. However, when the noise level is high, the high-capacity LSTM has higher errors than the regular LSTM. This is because the high-capacity LSTM has more trainable parameters and is more prone to over-fitting to the noise in the training labels. In contrast, the LSTM metamodel is more robust to label noise and is able to learn the true feature-label relationship better from the high-noise dataset. In practice, since the true relationship is unknown, the true error is not readily available. The test error approximates the true error and is directly observable. The differences in test errors of the LSTM metamodels are inconsistent with the differences in true errors. Therefore, test errors are not reliable indicators of the metamodels' noise tolerance. For practical applications that utilize nerual networks as metamodels, we recommend not to simulate the test scenarios. Instead, we recommend simulating the true contract losses on a subset of training scenarios to evaluate the metamodels' generalization ability.

In summary, our numerical experiments demonstrate that both LSTM metamodels are capable of learning the true inner simulation model from the noisy training labels. In most cases, a high-capacity metamodel is shown to perform better. Nevertheless, in extreme circumstances when the noise level is too high, using a high capacity metamodel leads to severe over-fitting and poor generalization. Hence, domain knowledge about the right architecture for the task and knowledge about the noise level in the training labels are beneficial for choosing the correct metamodel.

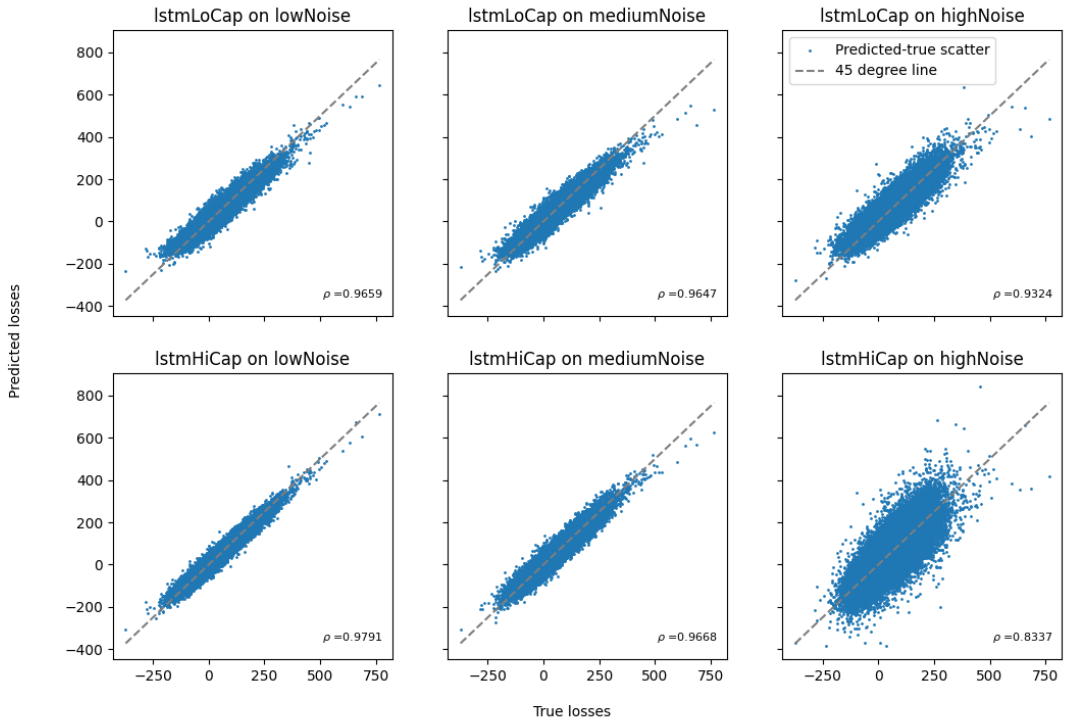


Figure 3.7: QQ-plots between true losses (x-axis) and predicted losses (y-axis) for two LSTM metamodels.

In further details, the QQ-plots depicted in Figure 3.7 illustrates the fit of the two LSTM metamodels when different noise levels are present in the training labels. They are arranged in a 2-by-3 grid, where the two rows correspond to the two LSTM metamodels with different model capacities, and the three columns correspond to the increasing noise levels. The Pearson correlation coefficient between the true losses and the predicted losses is labeled in each subplot. The sparsity of the plot increases as we move from left to right, indicating that the noise level in the training labels increases. There are two key findings.

1. When trained on the low-noise dataset, the metamodel predictions of high-capacity LSTM align more closely with the true labels than the regular LSTM. This is expected because the high-capacity LSTM has more trainable parameters and is able to learn more complex relationships better when the noise level is low.

2. When the noise level is medium or high, the high-capacity LSTM is more affected than the regular LSTM. This is because the high-capacity LSTM has more trainable parameters and is more prone to over-fitting to the noise in the training labels. In contrast, the regular LSTM is more robust to label noise.

To further test the sensitivity of a LSTM metamodel and provide a more comprehensive view of the noise tolerance of DNN models, we conduct a sensitivity analysis on the noise tolerance of the regular LSTM metamodel by varying the numbers of outer scenarios (M) and inner replications (N) used to generate the training dataset. The number of outer scenarios corresponds to the number of data points in the training dataset, and the number of inner replications controls the noise level in the training labels.

	$N = 1$	$N = 10$	$N = 100$	$N = 1000$
$M = 100$	1.139	0.229	0.167	0.158
$M = 1000$	0.559	0.173	0.123	0.127
$M = 10{,}000$	0.283	0.115	0.099	0.097
$M = 100{,}000$	0.129	0.070	0.063	0.063

Table 3.4: MSE between regular LSTM predicted losses and true losses.

	$N = 1$	$N = 10$	$N = 100$	$N = 1000$
$M = 100$	0.764	0.408	0.131	0.087
$M = 1000$	0.878	0.367	0.156	0.087
$M = 10{,}000$	0.351	0.147	0.064	0.063
$M = 100{,}000$	0.149	0.065	0.060	0.038

Table 3.5: MSE between high-capacity LSTM predicted losses and true losses.

Table 3.4 and 3.5 show the average squared errors between the LSTM predictions and the true losses for increasing M and N. The last rows of Table 3.4 and Table 3.5 show the performance of the regular and high capacity LSTM metamodels trained with $M = 100{,}000$ training labels with different noise levels. We observe that an increasing N reduces the MSE, but the reduction is not substantial for a regular LSTM when N is larger than 10. For the high-capacity LSTM, the MSE is substantially reduced when N is increased from 1 to 100, but the reduction is not substantial when N is increased from 100 to 1000. Another way to interpret the results in Table 3.4 is to compare the MSEs for the same budget of $\Gamma = M \times N$.

Table entries on the same off-diagonal have the same simulation budget Γ. For most budgets, the MSEs are also the lowest when $N = 10$. The results in Table 3.4 suggest that the performance of the LSTM metamodel is more sensitive to the number of outer scenarios than the number of inner replications. Treating the neural network as an advanced regression metamodel, we find this pheonomenon to be consistent with the results in Broadie et al. (2015), where the authors show that the performance of a regression-based nested simulation procedure is more affected by the number of outer scenarios. The last row in Table 3.4 and 3.5 show that the high-capacity LSTM metamodel is relatively insensitive to the number of inner replications. To make the most efficient use of a fixed simulation budget, the number of inner replication should be set constant around $N = 10$ while increasing the number of outer scenarios to the maximum.

To further investigate the LSTM metamodels' sensitivity to data quantity and quality (M and N, respectively), we report the Spearman rank correlation coefficients Table 3.6, which measure the ability to rank the scenarios by their true losses. It is an appropriate measure of the metamodel's performance in the two-stage procedure, where the metamodel is used to identify the predicted tail scenario set, on which extensive simulations are run in stage 2 to estimate the 95%-CVaR. The Pearson correlation coefficients are also included in the parentheses to illustrate the linear correlation between the predicted losses and the true losses. They measure the metamodel's overall prediction accuracy. Our previous numerical experiments illustrated in Figure 3.5 have shown that the LSTM metamodel predictions have high *Spearman* correlation with the true labels for the combination $M = 100{,}000$ and $N = 100$. We intend to examine if LSTM predictions also have high *Pearson* correlation with the true labels. A high *Pearson* correlation suggests the possibility of using LSTM predictions to estimate risk measures directly.

	$N = 1$	$N = 10$	$N = 100$	$N = 1000$
$M = 100$	0.638 (0.881)	0.875 (0.915)	0.896 (0.903)	0.937 (0.941)
$M = 1000$	0.722 (0.768)	0.899 (0.908)	0.886 (0.891)	0.922 (0.927)
$M = 10{,}000$	0.845 (0.630)	0.937 (0.900)	0.908 (0.905)	0.947 (0.948)
$M = 100{,}000$	0.935 (0.640)	0.963 (0.909)	0.927 (0.922)	0.965 (0.966)

Table 3.6: Spearman (Pearson) correlation coefficients of high-capacity LSTM predictions.

Consistent with our previous numerical experiments, the Spearman correlations of a high-capacity LSTM is high for a moderate simulation budget. For $M = 100{,}000$, the Spearman correlation is relativelyinsensitive to N. This finding supports further budget savings for our two-stage procedure. In stage 1, we can lower N from 100 to 10 almost without compromising on tail scenario preditions.

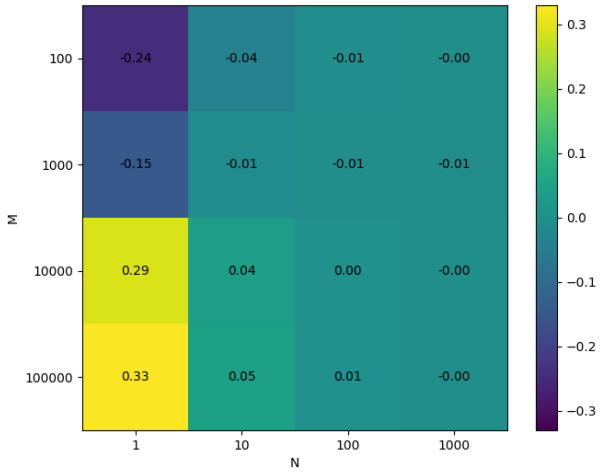


Figure 3.8: Difference between Spearman and Pearson correlations for high-capacity LSTM metamodel.

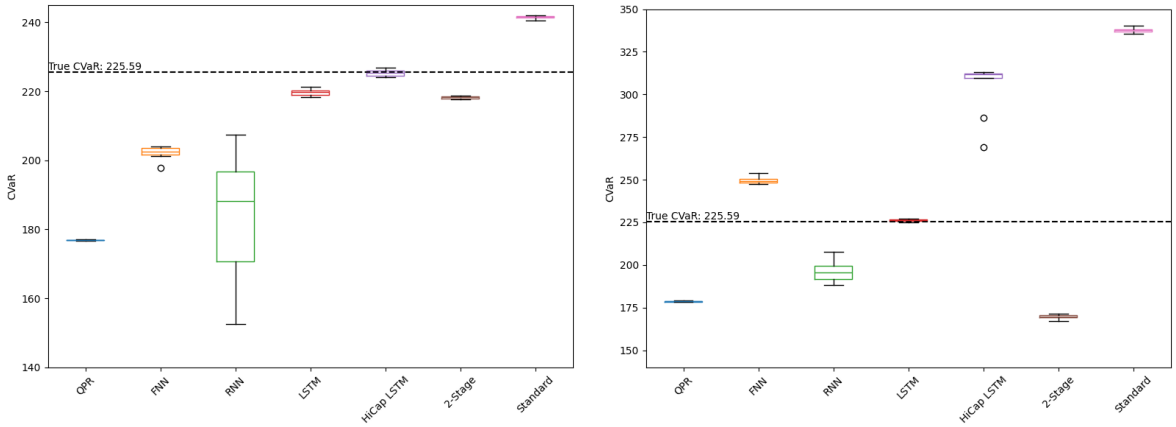
For an entry of Table 3.6, the total simulation budget $\Gamma = MN$. Entries on the same off-diagonal all have the same Γ. The Pearson correlation coefficients are substantially lower for $N = 1$ than the other N values.

Figure 3.8 shows the difference between the Spearman and Pearson correlation coefficients for the high-capacity LSTM metamodel. The heatmap is generated by subtracting the Pearson correlation coefficients from the Spearman correlation. We observe that Spearman correlation coefficients are not substantially different from the Pearson correlation coefficients for any N larger than 10. This is a strong evidence that the LSTM metamodel is not only able to effectively rank the scenarios by their true losses, but also able to make accurate loss predictions. Instead of using the LSTM metamodel only for classifying tail scenarios in a two-stage procedure, LSTM's ability to cut through a moderate level of noise in training labels encourages us to use its predictions to estimate the CVaR directly.

3.5.3 Single-Stage Procedure

The accuracy and robustness of the LSTM metamodels motivate us to propose a single-stage procedure that uses the metamodel predictions to estimate the CVaR directly. Instead of relying on the standard nested simulation procedure in stage 2, the single-stage

procedure can be even more efficient. In this section, we compare the single-stage procedure to the two-stage procedure and the standard procedure in estimating the 95%-CVaR of the hedging losses for GMWB. In our numerical experiments, the results for the single-stage procedure is obtained with the same metamodels as in the two-stage procedure. The only difference from Section 3.5.1 is that the metamodel predictions are used to estimate the risk measures directly.



(a) Medium noise training labels ($N = 10$) (b) High noise training labels ($N = 1$)

Figure 3.9: CVaR estimates of the single-stage procedure with metamodels.

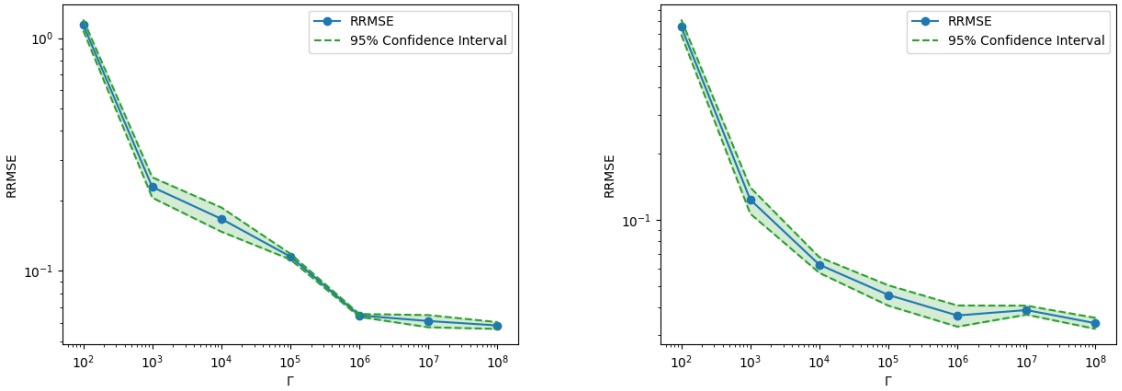
Figure 3.9 shows the boxplots of the 95%-CVaR estimates of the single-stage procedure with different metamodels and compares them to the two-stage procedure and the standard procedure. The two-stage procedure uses the same simulation budget as the single-stage procedure in stage 1 with an extra amount of budget for an extensive inner simulation on the predicted tail scenarios in stage 2. The metamodel with the best performance in the two-stage procedure is selected, and the safety margin is set to 0%. The standard procedure shown in Figure 3.9 uses the noisy loss labels in the training dataset to estimate the CVaR directly, which uses the same simulation budget as the single-stage procedure. We observe that the single-stage procedures with the regular LSTM metamodel consistently produce CVaR estimates that are closer to the true value than the standard procedure's estimate. This finding is especially impressive when the noise level in the training labels is high. It is another strong evidence that a well-selected LSTM metamodel is able to cut through the noise in the noisy training labels and make accurate loss predictions that lead to accurate CVaR estimates. The difference in performance among the metamodels is more pronounced in the single-stage procedure than in the two-stage procedure. Trained using medium noise labels, the high-capacity LSTM metamodel consistently produces CVaR estimates that are

closer to the true value than the regular LSTM, while the other metamodels produce worse estimates.

However, when the noise level in the training labels is high, the high-capacity LSTM suffers from severe overfitting. This is also evident in Table 3.3. Since we are using the metamodel predictions to estimate the CVaR directly without any safety margin, the metamodel's overall prediction accuracy becomes more important. In other words, the single-stage procedure is more sensitive to the metamodel's ability to make accurate loss predictions than the two-stage procedure. Previously, a generic FNN metamodel performs well in the two-stage procedure. However, results in Figure 3.9 suggest that the FNN metamodel should not be used in the single-stage procedure.

A single-stage procedure with a regular LSTM metamodel is particularly superior to the two-stage procedure, as it is able to achieve a higher accuracy as the standard procedure with a much smaller amount of simulation budget. Note that the single-stage procedure does not require a safety margin. By avoiding the calibration of the safety margin, it is more straight-forward to implement and is more efficient than the two-stage procedure. For a two-stage procedure with a 0% safety margin, the extra computational cost is from the extensive inner simulation in stage 2. On 4 20-core Intel Xeon Gold 6230 processors, stage 2 of the two-stage procedure takes 30 minutes to run with parallel processing, while the 95% confidence band of training time of the high-capacity LSTM metamodel is (19.64 ± 0.94) minutes on an Nvidia RTX 3060 Ti GPU. While the accuracy of the two-stage procedure is highly dependent on the safety margin, increasing the safety margin introduces extra computational cost. Therefore, while achieving a higher accuracy in estimating the CVaR, the single-stage procedure requires only 60% computation time of the two-stage procedure.

To further investigate the single-stage procedure's performance, we conduct a convergence analysis on the LSTM metamodels. Figure 3.10 shows the log-log plots of the Relative Root Mean Squared Error (RRMSE) between the LSTM metamodel CVaR predictions and the true CVaR against the total simulation budget Γ. For each budget Γ, the metamodels are trained for different combinations of M and N, and the metamodel with the best RRMSE is plotted. While numerical results suggest that the CVaR estimator of the single-stage procedure with LSTM metamodels may have better accuracy than the two-stage procedure and the standard procedure when the number of outer scenarios is further increased, we found it difficult to compare their performance due to insufficient computational resources. For $\Gamma \leq 100{,}000$, the number of outer scenarios is fixed at $M = 100{,}000$, and only the number of inner replications is varied. Instead, we try to analyze the effect of the number of outer scenarios and the number of inner replications separately by fixing one and varying the other.



(a) Regular LSTM (b) High-capacity LSTM

Figure 3.10: Empirical convergence of CVaR for the single-stage procedure with LSTM metamodels.

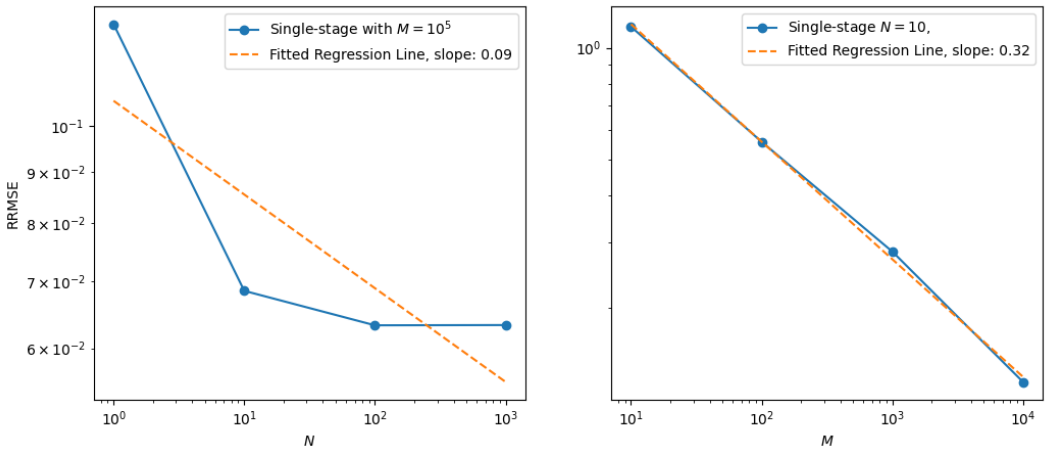


Figure 3.11: Empirical convergence of the single-stage procedure with a LSTM metamodel.

Figure 3.11 shows the log-log plots of the RRMSE between the LSTM metamodel CVaR predictions and the true CVaR against the simulation budget. The left figure shows the empirical convergence of the RRMSE for increasing inner replications with a fixed number of outer scenarios ($M = 100{,}000$), and the right figure shows the empirical convergence of the RRMSE for increasing outer scenarios with a fixed number of inner replications ($N =$

10). Due to computational constraints, we are not able to run the two-stage procedure and the standard procedure with more outer scenarios or more inner replications, therefore the comparison between the single-stage procedure is only available up to $M = 100{,}000$ and $N = 1000$. We observe that the RRMSE decreases as the simulation budget increases, and the rate of convergence is higher when the quantity of the data increases. For an LSTM metamodel, increasing the data quality of the training labels has diminishing returns, which is consistent with the results in Figure 3.11. For an increasing number of inner replications with a fixed number of outer scenarios, the RRMSE ceases to decrease after reaching $N = 100$. More specfically, when the quality of the training labels is fixed at $N = 10$, the CVaR estimator of a single-stage procedure with a LSTM metamodel converge roughly in the order of $\mathcal{O}(M^{\frac{2}{3}})$. This observation resonates with the best possible convergence rate of a standard nested simulation procedure in Gordy and Juneja (2010). Hence, in practice, we suggest running the single-stage procedure with a moderate number of inner replications and a large number of outer scenarios to achieve a high level of accuracy with a reasonable computational cost.

3.6 Conclusion

The proposed nested simulation procedures with DNN metamodels are shown to result in substantial computational savings in estimating CVaR of the hedging loss of a VA contract from accurately predicting the hedging losses and identifying the tail scenarios. When new outer scenarios are generated, a trained LSTM metamodel can distinguish between tail and non-tail scenarios and make accurate predictions without the need to run new inner simulations.

Our novel experiment design allows us to examine the impact of label noise on DNN models. We find that a DNN with a suitable architecture is able to cut through the noise in training labels and learn the true complex dynamic hedging model. By showcasing the resilience of these models, our study aims at encouraging regulatory bodies to recognize the value and applicability of deep learning metamodels in financial risk management, and it provides informed suggestions and guidance for the incorporation and oversight of advanced deep learning metamodel in Monte Carlo Simulation in financial applications. Our findings are particularly insightful in this context.

In our numerical experiments, a LSTM metamodel is resilient to a high level of noise in training labels and is able to make accurate predictions. This is an encouraging evidence that DNN metamodels can be used to improve the efficiency of Monte Carlo simulation for quantitative risk management tasks that require a computational-expensive simulation

procedure. We propose two nested simulation procedures that use DNN metamodels to estimate risk measures of the hedging loss of a VA contract. For estimating tail risk measures, our two-stage procedure is designed to address regulatory concerns by avoiding the direct use of metamodel predictions but instead by using them to identify the potential tail scenarios. An extensive inner simulation is performed in this chapter to achieve a high level of accuracy on the predicted tail scenarios. However, the safety margin in the two-stage procedure is a user's choice and is not easy to determine before running extensive numerical experiments.

Our single-stage procedure uses the metamodel predictions to estimate the risk measure directly. It is more efficient and can be extended to estimate risk measures that require knowledge of the entire loss distribution. Our numerical experiments demonstrate that the proposed single-stage procedure with DNN metamodels result in further computational savings over our two-stage procedure. Furthermore, our numerical results provide evidences for adopting DNN metamodels in Monte Carlo simulation for risk management tasks. Through our systematic empirical study of the noise tolerance of DNN metamodels, we address regulatory concerns by showing that a LSTM metamodel with moderate model capacity is resilient to a high level of noise in training labels and is able to make accurate predictions. A possible future research direction is to apply DNN metamodels in other financial risk management tasks that requires complex nested simulation with high-dimensional outer scenarios. Another future research direction is to investigate the impact of label noise on other deep learning models, such as convolutional neural networks and transformer models, and to compare their performance with LSTM metamodels in nested simulation procedures. From a practical standpoint, the choice of a suitable DNN architecture is crucial for the success of a deep learning metamodel in nested simulation procedures. We find that a LSTM metamodel is the most suitable for our dynamic hedging simulation model with time series features, but the optimal network architecture may vary for different simulation models. Exploring DNN metamodels in other complex risk management tasks presents a promising avenue for research, especially as these tasks often involve complex, high-dimensional scenarios where traditional methods are insufficient. The versatility of neural networks could unlock new insights across a broad spectrum of financial and actuarial applications.

In this chapter, we have demonstrated the potential of DNN metamodels to estimate risk measures with high accuracy and efficiency when simulation data is abundant. In practice, simulation data is often scarce for new market conditions and new insurance products. DNN metamodels are known to suffer from over-fitting when the number of training samples is limited. In the next chapter, we will extend our study to transfer learning and show that DNN metamodels trained on an existing VA contract can be transferred to a new

VA contract with different market conditions. We hypothesize that the DNN metamodel will generalize well to the new VA contract, and the computational savings from transfer learning will be substantial.

Chapter 4

Transfer Learning for Rapid Adaptation of Deep Neural Network Metamodels in Dynamic Hedging of Variable Annuities

4.1 Introduction

In the evolving landscape of financial markets, insurance products such as VAs have gained a substantial amount of interest due to their ability to provide both investment growth and guaranteed benefits. Managing risks associated with these products in volatile market conditions is a complex task that demands sophisticated financial modeling techniques such as nested simulation. A standard nested simulation procedure involves running a large number of simulations to generate a dataset of scenario-wise contract losses with two levels of simulations. Practitioners can estimate tail risk measures using the realized losses for different risk scenarios obtained from these simulations. Nested simulation is extremely computationally expensive. To address the computational burden, metamodeling techniques have been proposed, where a surrogate model approximates the outcomes of a complex simulation procedure. Traditional Machine Learning (ML) models often struggle to capture the intricate nonlinear relationships and temporal dependencies inherent in financial data. In particular, DNNs, and specifically LSTM networks, have been employed as metamodels to predict the outcomes of the inner simulations efficiently. Chapter 3 introduced a metamodeling-based nested simulation framework for dynamic hedging of VAs

that approximates the inner simulation by an LSTM metamodel. It shows that LSTMs, as a type of gated RNNs, were well-suited as metamodels for MC simulation of financial time series and could capture the long-term dependencies in the simulation dataset. This property is crucial for the application of dynamic hedging of VAs.

Despite the advantages of using DNN metamodels, computational challenges arise when market conditions change or new VA contracts with different features are introduced. Retraining neural network metamodels from the scratch in response to every change is computationally inefficient and time-consuming. Practitioners with limited resources face several challenges:

1. The simulation budget is limited, but DNN metamodels require extensive training data to learn the feature-label relationship (Golestaneh et al., 2024).

2. The computational cost of retraining a suitable DNN metamodel is often much higher than fitting a traditional parametric regression.

3. Financial markets are dynamic with frequent shifts in volatility, interest rates, and other risk factors (Cont, 2001).

Constraints on the simulation budget is the most critical challenge for dynamic hedging of VAs. Any method that involves nested simulation is computationally expensive, and the computational cost is often prohibitive for changing market conditions. Therefore, it is essential to develop methods that can rapidly adapt existing metamodels to new conditions without incurring the full computational cost of retraining and extensive addtional simulations. Transfer Learning (TL) offers a compelling solution to this problem by enabling the reuse of a pre-trained model on a new but related task (Pan and Yang, 2009). In the context of DNNs, TL involves leveraging the knowledge acquired during training on one dataset to improve learning performance on a different dataset (Yosinski et al., 2014). This approach can substantially reduce the amount of training data needed, training time and computational resources while enhancing model generalization. Instead of starting from the scratch, a new DNN metamodel can be built on top of a pre-trained metamodel and fine-tuned on new data. This allows the metamodel to adapt quickly to unseen conditions and new VA contracts.

In this chapter, we explore the application of TL to the dynamic hedging of VAs using LSTM metamodels. We propose a novel TL framework that accelerates the training of DNN metamodels for nested simulation procedures in dynamic hedging. This setting is particularly relevant for dynamic hedging problems, as insurance companies often underwrite new contracts under changing market conditions and actuarial assumptions. This

chapter proposes a TL framework that makes fast adaptation of metamodels possible. Our approach involves pre-training an LSTM network on a large dataset of VA simulations and then fine-tuning it on a smaller dataset of new VA contracts. We evaluate the performance of the TL framework on a real-world dataset of VA contracts and compare it with training a LSTM metamodel from scratch. More importantly, we show the strengths and weaknesses of different TL techniques for rapid adaptation of LSTM metamodels.

The rest of this chapter is organized as follows. Section 4.2 provides an overview of the dynamic hedging problem for VAs and the use of LSTM networks as metamodels in a nested simulation procedure. Section 4.2.4 introduces the TL framework for rapid adaptation of LSTM metamodels in dynamic hedging. Section 4.3 presents the experimental setup and results, comparing the performance of TL with training from scratch. Finally, Section 4.4 concludes the chapter and discusses future research directions.

4.2 Transfer Learning in Financial Metamodeling

TL is a machine learning paradigm where knowledge acquired from a source task is utilized to improve learning performance on a related target task. One of the primary applications of TL in finance is asset price prediction. Traditional models, such as geometric Brownian motion (GBM), autoregressive integrated moving average (Autoregressive Integrated Moving Average (ARIMA)) and generalized autoregressive conditional heteroskedasticity (Generalized Autoregressive Conditional Heteroskedasticity (GARCH)), have been widely used in time series modeling. ARIMA models (Box and Pierce, 1970) combine autoregression, differencing, and moving average components to capture linear relationships in time series data. GARCH models (Bollerslev, 1990), extend Autoregressive Conditional Heteroskedasticity (ARCH) models to better capture volatility clustering, which is a key stylized fact of financial time series (Cont, 2001). However, these models often struggle to capture complex dynamics, e.g., regime changes (Hamilton, 1989) and extreme market events (Embrechts et al., 2013) that are prevalent in financial markets. In Chapter 3, our numerical experiment uses a regime-switching geometric Brownian motion (RS-GBM) as the underlying asset model. For estimating tail risk measures, a metamodel-based nested simulation procedure is proposed to approximate the inner simulation. We have shown DNN metamodels like RNN and LSTM networks have demonstrated substantial improvements in modeling temporal dependencies, and they save a substantial amount of simulation budget. However, training these models from the scratch requires an extensive amount of simulation data, which may not always be available for specific assets or under certain market conditions.

TL offers a solution to this problem by leveraging knowledge from related assets or tasks to improve the learning performance of DNN metamodels on the target task. In algorithmic trading, Jeong and Kim (2019) used TL to enhance the performance of a reinforcement learning agent by preventing overfitting from an insufficient amount of market data. TL techniques have also been applied in building fraud detection systems. Financial fraud often exhibits subtle and evolving patterns, and it is challenging to develop robust detection models. By transferring previous knowledge from detected fraud cases, models can adapt to detect new fraud schemes more effectively (Lebichot et al., 2021). Yan et al. (2024) conduct a comprehensive survey study of current TL techniques in financial applications, and they find almost all applications of TL only employ parameter transfer, where the pre-trained model is fine-tuned on the target task. Our multi-task learning framework in Section 4.2.4 extends beyond parameter transfer to shared representation learning across multiple tasks.

Regarding a nested simulation of VAs, Cheng et al. (2019) is the most relevant study to our work, where they proposed a TL framework for fast valuation of large portfolios of VAs. Instead of using stochastic kriging (Gan and Lin, 2015), they employed a pre-trained DNN to select the best representative scenarios from a large portfolio of VAs. In our work, we focus on the application of TL to accelerate the training of LSTM metamodels for a nested simulation in dynamic hedging of VAs. The fine-tuned LSTM metamodels can be readily adapted to the two-stage procedure and the single-stage procedure in Chapter 3 to predict the contract losses under different scenarios.

In our context of DNNs metamodeling-based simulation procedures for hedging VAs, TL involves pre-training a DNN metamodel where the simulation budget is abundant and then fine-tuning it on a smaller dataset of new contracts or market conditions. Written on the same underlying asset, different VA contracts may share common features or exhibit similar patterns, especially temporal dependencies and regime changes in the underlying financial time series. Similarly, two VAs with the same features but based on different underlying assets may have some shared characteristics that can be leveraged to improve the learning performance of the metamodel. By transferring knowledge from a pre-trained model to a new but related task, TL can substantially reduce the computational cost of training the metamodel on the target task.

LSTM networks are well-suited for modeling sequential data due to their ability to capture long-term dependencies (Hochreiter and Schmidhuber, 1997). For the application of VA risk management using metamodel-based nested simulation, LSTM networks approximate the inner simulation, i.e., the mapping from scenarios to the scenario-wise contract losses. In Chapter 3, we treat metamodeling as a supervised learning problem and demonstrated that LSTM networks could effectively model this complex relationships

with extensive training on a large dataset of VA simulations. The total computational cost originates from two sources: running the standard nested simulation procedure to generate the training data and training the LSTM network on the generate dataset. Given a new VA contract, these steps need to be repeated to adapt the LSTM metamodel to new market conditions and contract features. For a DNN with a large number of parameters, retraining the LSTM network from the scratch can be computationally expensive. Generating a large training dataset using the standard nested simulation procedure is costly, and the computational burden can be prohibitive for certain assets or market conditions. TL offers an alternative approach that keeps the previous knowledge, i.e., the pre-trained model, as a foundation. Building on the pre-trained model, the LSTM network can be fine-tuned on a smaller dataset of new contracts or market conditions. This approach accelerates the adaptation of the LSTM metamodel to new conditions. Computational savings come in two forms:

1. fine-tuning requires less training time than training LSTM metamodels from scratch, and
2. fewer training data points are needed to achieve a good LSTM metamodel, which is particularly beneficial when the standard nested simulation procedure is costly to implement.

In supervised learning, a **domain** $\mathcal{D}$ comprises a feature space $\mathcal{X}$ and a marginal probability distribution F. Correspondingly, a **task** $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a predictive function $f : \mathcal{X} \to \mathcal{Y}$ that maps input features to output labels.

A typical TL framework for supervised learning consists of the following:

- **source domain**: $\mathcal{D}_{\text{So}} = \{\mathcal{X}_{\text{So}}, F_{\text{So}}(\mathbf{X})\}$,
- **source task**: $\mathcal{T}_{\text{So}} = \{\mathcal{Y}_{\text{So}}, f_{\text{So}}(\cdot)\}$,
- **target domain**: $\mathcal{D}_{\text{Ta}} = \{\mathcal{X}_{\text{Ta}}, F_{\text{Ta}}(\cdot)\}$,
- **target task**: $\mathcal{T}_{\text{Ta}} = \{\mathcal{Y}_{\text{Ta}}, f_{\text{Ta}}(\cdot)\}$,

where $\mathcal{X}_{\text{So}}$ and $\mathcal{X}_{\text{Ta}}$ include input features derived from the outer simulation of two different VA contracts or market conditions, and F_{So} and F_{Ta} are the marginal probability distributions of the source and target domains, respectively. Hence, the source domain and task are related to one VA contract for a certain market condition, while the target domain

and task are related to new VA contracts and market conditions for which we don't have the simulation budget to run the standard nested simulation procedure extensively. In our numerical experiments, the dataset is generated by running the standard nested simulation procedure in Algorithm 1 for a large number of VA contracts. The input features $\mathbf{X}$ are the return vectors from Equation (3.11), and the output labels L are the contract losses for different outer scenarios. The stock price vector $\mathbf{S}$ is not crucial for the discussion of this chapter and is omitted since it can be immediately recovered from $\mathbf{X}$ and the initial stock price S_0. We keep the number of features to be 240 for both source and target domains as in Section 3.5. The predictive function f_{So} and f_{Ta} are trained to estimate outputs L_{So} and L_{Ta}, i.e., approximating the inner simulations under the source and target tasks, respectively. TL seeks to improve the learning of the target predictive function $f_{\text{Ta}}(\cdot)$ in $\mathcal{D}_{\text{Ta}}$ by leveraging the previous training on $\mathcal{D}_{\text{So}}$ and $f_{\text{So}}(\cdot)$, particularly when $\mathcal{D}_{\text{So}} \neq \mathcal{D}_{\text{Ta}}$ or $\mathcal{T}_{\text{So}} \neq \mathcal{T}_{\text{Ta}}$. In Chapter 3, we have $\mathcal{D}_{\text{So}} = \mathcal{D}_{\text{Ta}}$ and $\mathcal{T}_{\text{So}} = \mathcal{T}_{\text{Ta}}$.

In our TL setting, both the source and target tasks involve learning a mapping from the risk factors to the VA contract losses. For the source task, we have a dataset $\mathcal{D}_{\text{So}} = (X_{\text{So}}^{(i)}, L_{\text{So}}^{(i)})_{i=1}^{M_{\text{So}}}$, where M_{So} is the number of training samples, i.e., the number of outer simulation paths used to generate the dataset with a standard nested simulation procedure. $X_{\text{So}}^{(i)} \in \mathcal{X}_{\text{So}}$ and $L_{\text{So}}^{(i)} \in \mathcal{Y}_{\text{So}}$. The ultimate objective is to learn a metamodel f_{Ta} that predicts the VA contract losses L_{So} from the scenarios X_{Ta} in the form of a financial time series. TL starts by training a metamodel f_{So} on the source domain $\mathcal{D}_{\text{So}}$. For both the two-stage procedure and the single-stage procedure in Chapter 3, f_{So} is approximated by an LSTM network $f_{\text{So}}(\cdot;\theta_{\text{So}})$, where the parameters of the LSTM network, θ_{So} are learned by minimizing an MSE loss function on the source domain. Then the pre-trained parameters θ_{So} to inform the learning of $f_{\text{Ta}}(\cdot;\theta_{\text{Ta}})$ on the target domain $\mathcal{D}_{\text{Ta}}$. The training on the target domain should encourage similarity between θ_{So} and θ_{Ta} to facilitate the transfer of knowledge.

The most common TL techniques for supervised learning include fine-tuning, layer freezing, and multi-task learning. These techniques can be categorized based on how they leverage pre-trained metamodels and how they encourage the similarity between $f_{\text{So}}(\cdot;\theta_{\text{So}})$ and $f_{\text{Ta}}(\cdot;\theta_{\text{Ta}})$.

4.2.1 Fine-tuning

Fine-tuning is a commonly used TL technique that uses the **same neural network architecture** for both the source and target tasks. We first train the source LSTM metamodel $f_{\text{So}}(\cdot;\theta_{\text{So}})$ on $\mathcal{D}_{\text{So}}$, capturing temporal dependencies and patterns relevant to the source

task. For the target task, we initialize $\theta_{\text{Ta}} = \theta_{\text{So}}$ and proceed to fine-tune the entire network on $\mathcal{D}_{\text{Ta}}$ by using a smaller learning rate.

Algorithm 4 Fine-tuning Metamodel for a Target Task

1: **Input:** source dataset $\mathcal{D}_{\text{So}} = \{(X_{\text{So}}^{(i)}, L_{\text{So}}^{(i)})\}_{i=1}^{M_{\text{So}}}$, target dataset $\mathcal{D}_{\text{Ta}} = \{(X_{\text{Ta}}^{(i)}, L_{\text{Ta}}^{(i)})\}_{i=1}^{M_{\text{Ta}}}$, learning rate α_{So} and smaller learning rate α_{Ta}.

2: Train a LSTM metamodel $f_{\text{So}}(\cdot; \theta_{\text{So}})$ on $\mathcal{D}_{\text{So}}$:

$$\theta_{\text{So}} = \min_{\theta} \frac{1}{M_{\text{So}}} \sum_{i=1}^{M_{\text{So}}} \left(f_{\text{So}}(X_{\text{So}}^{(i)}; \theta) - L_{\text{So}}^{(i)} \right)^2. \tag{4.1}$$

3: Initialize the target metamodel parameters θ_{Ta} using the pre-trained metamodel parameters:

$$\theta_{\text{Ta}} \leftarrow \theta_{\text{So}}.$$

4: Fine-tune the entire LSTM model $f_{\text{Ta}}(\cdot; \theta_{\text{Ta}})$ on the target dataset $\mathcal{D}_{\text{Ta}}$ using a smaller learning rate α_{Ta}:

$$\theta_{\text{Ta}} = \min_{\theta} \frac{1}{M_{\text{Ta}}} \sum_{i=1}^{M_{\text{Ta}}} \left(f_{\text{Ta}}(X_{\text{Ta}}^{(i)}; \theta) - L_{\text{Ta}}^{(i)} \right)^2. \tag{4.2}$$

5: **Output:** Final adapted LSTM metamodel $f_{\text{Ta}}(\cdot; \theta_{\text{Ta}})$ for the target task.

Algorithm 4 assumes that learned sequential representations are beneficial for the target task. Only minor adjustments are needed to adapt the metamodel to the new task, and the training process is accelerated by the pre-trained parameters. Fine-tuning is particularly useful when the nested simulation procedures for the two VA contracts are closely related. Only minor adjustments are needed to adapt the LSTM metamodel to the new domain.

4.2.2 Layer Freezing

In a layer freezing approach, we partition the model parameters into frozen parameters θ_0 and trainable parameters θ_1, such that $\theta = [\theta_0, \theta_1]$. Typically, θ_0 are parameters of the lower layers, and θ_1 are parameters of the higher layers that include the output layer. The intuition behind layer freezing is that the lower layers capture general features that are transferable across tasks, while the higher layers are more task-specific.

Algorithm 5 Layer Freezing for Metamodel Transfer

1: **Input:** source dataset $\mathcal{D}_{\text{So}} = \{(X_{\text{So}}^{(i)}, L_{\text{So}}^{(i)})\}_{i=1}^{M_{\text{So}}}$, target dataset $\mathcal{D}_{\text{Ta}} = \{(X_{\text{Ta}}^{(i)}, L_{\text{Ta}}^{(i)})\}_{i=1}^{M_{\text{Ta}}}$, learning rates α_{So} and α_{Ta}, frozen parameters θ_0, trainable parameters θ_1.

2: Train LSTM model $f_{\text{So}}(\cdot; \theta_{\text{So}})$ on $\mathcal{D}_{\text{So}}$:

$$\theta_{\text{So}} = [\theta_0, \theta_1] = \min_{\theta} \frac{1}{M_{\text{So}}} \sum_{i=1}^{M_{\text{So}}} \left(f_{\text{So}}(X_{\text{So}}^{(i)}; \theta) - L_{\text{So}}^{(i)} \right)^2. \tag{4.3}$$

3: Initialize the target model parameters $\theta_{\text{Ta}} = [\theta_0, \theta_1]$ using the pre-trained source model parameters θ_{So}:

$$\theta_{\text{Ta}} \leftarrow \theta_{\text{So}} = [\theta_0, \theta_1].$$

4: Freeze the parameters of the shared layers θ_0:

5: Fine-tune the trainable layers θ_1 on the target dataset $\mathcal{D}_{\text{Ta}}$ using Algorithm 4:

$$\theta_{\text{Ta}} = \min_{\theta_1} \frac{1}{M_{\text{Ta}}} \sum_{i=1}^{M_{\text{Ta}}} \left(f_{\text{Ta}}(X_{\text{Ta}}^{(i)}; [\theta_0, \theta_1]) - L_{\text{Ta}}^{(i)} \right)^2. \tag{4.4}$$

6: **Output:** Final adapted LSTM metamodel $f_{\text{Ta}}(\cdot; [\theta_0, \theta_1])$ for the target task.

4.2.3 Multi-task Learning

Multi-task learning (Caruana, 1997) refers to a machine learning paradigm where a single model is trained simultaneously on multiple related tasks. Shared representations are learned across tasks, which can improve learning efficiency and predictive performance on each individual task with limited data. In contrast to fine-tuning and layer freezing, multi-task learning aims at leveraging learned knowledge from multiple tasks, and all tasks are trained simultaneously.

Let $\{\mathcal{T}_k\}_{k=1}^{K}$ represent a set of K related tasks, each corresponding to a metamodeling task for a different standard nested simulation procedure. For each task $\mathcal{T}_k$, we have a dataset $\mathcal{D}_k = (X_k^{(i)}, L_k^{(i)})i = 1^{M_k}$, where M_k is the number of training samples for task k. $X_k^{(i)}$ and $L_k^{(i)}$ are the features and contract loss labels for task k, respectively.

For the implementation of our metamodeling studied in Chapter 3, we consider a multi-task learning framework where the LSTM layers are shared across multiple tasks, and each task has its own fully connected layer for prediction. The network parameters are divided into shared parameters θ_0 (LSTM layers) and task-specific parameters θ_k (fully connected

layers for task k). As opposed to layer freezing, all parameters are trainable.

The objective function for multi-task learning is the sum of the loss functions of all tasks:

$$\min_{\theta_0,\theta_1,\ldots,\theta_K} = \sum_{k=1}^{K} \frac{1}{M_k} \sum_{i=1}^{M_k} \left(f_i(X_k^{(i)}; \theta_0, \theta_1, \ldots, \theta_K) - L_k^{(i)} \right)^2, \tag{4.5}$$

where MSE loss function is used as the error metric, and $f_i(\cdot; \theta_0, \theta_1, \ldots, \theta_K)$ is the output of the network for task k. In essence, multi-task learning uses a multi-head architecture, where each task has its own output head, but the shared LSTM layers learn a common representation across tasks. Transfer occurs through the shared LSTM layers θ_0. These layers learn representations of temporal patterns and dependencies common to all tasks, effectively **pooling** information from multiple simulation schemes. The task-specific fully connected layers $[\theta_1, \ldots, \theta_K]$ allow each task to capture unique characteristics not shared with other tasks.

Algorithm 6 Multi-task Learning Framework for LSTM Metamodels

1: **Input:** learning rate α, set of K tasks $\{\mathcal{T}_k\}_{k=1}^K$ with datasets $\mathcal{D}_k = \{(X_k^{(i)}, L_k^{(i)})\}_{i=1}^{M_k}$, task-specific parameters θ_k for each task k, and shared parameters θ_0.

2: Train the multi-head LSTM metamodel on all K tasks simultaneously by minimizing the multi-task loss function:

$$\min_{\theta_0, \{\theta_k\}_{k=1}^K} \sum_{k=1}^{K} \frac{1}{M_k} \sum_{i=1}^{M_k} \left(f_i(X_k^{(i)}; \theta_0, \theta_k) - L_k^{(i)} \right)^2. \tag{4.6}$$

3: Update both the shared parameters θ_0 and task-specific parameters $\{\theta_k\}_{k=1}^K$ simultaneously using backpropagation and gradient descent with learning rate α.

4: **Output:** Trained multi-task LSTM metamodel $f(\cdot; \theta_0, \{\theta_k\}_{k=1}^K)$ for all K tasks.

Algorithm 6 outlines the multi-task learning framework for LSTM metamodels for nested simulation procedures. In Chapter 2 and Chapter 3, we demonstrated that pooling using a metamodel could improve the computation efficiency and estimator accuracy in nested simulation procedures. Here, pooling happens at a higher level, where information from multiple metamodels is shared to improve the learning performance on individual tasks. The multi-task loss function encourages the shared LSTM layers to learn generalizable representations of temporal dependencies that are beneficial across all tasks.

Choosing appropriate simulation schemes is crucial for the success of multi-task learning. The tasks should be related to ensure that the shared layers can capture common features beneficial across tasks. Criteria for selecting tasks with similar simulation schemes include:

- **Similarity in contract specifications:** simulations involving VA contracts with similar features are likely to share underlying risk factors and policyholder behaviors.
- **Similarity in underlying assets:** datasets simulated under different but related asset models can provide diverse information that enrich the shared representations. Datasets simulated under extreme market conditions can help the shared layers learn to hedge against tail risks.
- **Temporal Dynamics:** VA contracts with comparable maturity and rebalance frequency can help the shared layers learn temporal dependencies consistent across tasks.

Suppose that we aim at developing an LSTM metamodel for a GMWB contract under a stochastic volatility asset model, but we have limited data for this simulation scheme. For multi-task learning, we can select related nested simulation procedures with relatively abundant data, such as:

- GMMB contracts under a stochastic volatility asset model.
- GMWB contracts under a Black-Scholes asset model.
- GMWB contracts under a stochastic volatility model with different simulation parameters.

These tasks need to share similarities in contract features and market dynamics so that the shared LSTM layers can learn relevant temporal patterns applicable to the target task. Otherwise, the shared layers may not be able to learn generalizable representations of temporal dependencies, and the multi-task learning framework may lead to negative transfer. Negative transfer refers to the phenomenon where the performance of the target task is worse than that of training directly on the target task without TL. It occurs when the tasks are not related or an improper TL technique is used. Positive transfer, on the other hand, refers to the phenomenon where the performance of the target task is better than that of training directly on the target task without TL. Training a multi-task LSTM

metamodel benefits from having more training data, and the shared layers can learn more generalizable representations of temporal dependencies. Furthermore, if a new but similar task is introduced, fine-tuning, layer freezing, and multi-task learning can be combined to leverage the pre-trained model effectively. More specfically, the shared layers can be frozen, and the task-specific layers for a similar task in the training set can be fine-tuned on the new task. This shared knowledge helps the model generalize better on a target task, as the LSTM layers have been exposed to a wider variety of patterns and dynamics.

4.2.4 Rapid Adaptation of LSTM Metamodels

In this section, we propose a TL framework for rapid adaptation of LSTM metamodels in dynamic hedging of VAs. The goal is to leverage the knowledge acquired during training on a large dataset of VA simulations to improve the learning performance on a smaller dataset of new VA contracts.

Algorithm 7 combines the strengths of Multi-task learning, layer freezing, and fine-tuning to accelerate the training of LSTM metamodels.

- **Multi-task learning** allows the shared LSTM layers to pool information across related tasks, and it allows the metamodel to learn generalizable representations of temporal dependencies.
- **Layer freezing** ensures that shared features learned from the source tasks are retained when adapting to a new task, reducing the risk of overfitting to the target simulation data.
- **Fine-tuning** enables the task-specific layers to adapt quickly to the new task with a minimum amount of simulation cost, leveraging the pre-trained shared layers.

The combination of these techniques significantly reduces the computational cost of training LSTM metamodels for a new but related VA contract.

4.3 Numerical Experiments

In this section, we evaluate the performance of the TL framework for rapid adaptation of LSTM metamodels in dynamic hedging of VAs. The low noise dataset generated by the standard nested simulation procedure in Section 3.5 is used to train the source LSTM metamodels. We consider TL to two types of target tasks:

Algorithm 7 Transfer Learning Framework for LSTM Metamodels: Combining Fine-tuning, Layer Freezing, and Multi-task Learning

1: **Input:** Set of K tasks $\{\mathcal{T}_k\}_{k=1}^{K}$, with datasets $\mathcal{D}_k = \{(X_k^{(i)}, L_k^{(i)})\}_{i=1}^{M_k}$ for each task k, target dataset $\mathcal{D}_{\text{Ta}}$, learning rate α_{So} and α_{Ta}, shared parameters θ_0, task-specific parameters θ_k for each task k.

2: **Multi-task learning:** define shared LSTM layers θ_0 and task-specific fully connected layers θ_k for each task k. The LSTM layers are shared across all tasks $\{\mathcal{T}_k\}_{k=1}^{K}$.

3: Train the multi-task LSTM metamodel on all K tasks simultaneously:

$$\min_{\theta_0, \{\theta_k\}_{k=1}^{K}} \sum_{k=1}^{K} \frac{1}{M_k} \sum_{i=1}^{M_k} \left(f_i(X_k^{(i)}; \theta_0, \theta_k) - L_k^{(i)} \right)^2. \tag{4.7}$$

4: For a related task of hedging a new VA contract, combine fine-tuning and layer freezing:

5: **Layer freezing:** once the multi-task metamodel is trained, freeze the parameters of the shared LSTM layers θ_0.

6: **Fine-tuning:** based on task similarity, initialize the target task model parameters θ_{Ta} using parameters θ_k from task k.

$$\theta_{\text{Ta}} \leftarrow \theta_k$$

7: Fine-tune only θ_{Ta} on the new target dataset $\mathcal{D}_{\text{Ta}}$ using a smaller learning rate α_{Ta}:

$$\min_{\theta_{\text{Ta}}} \frac{1}{M_{\text{Ta}}} \sum_{i=1}^{M_{\text{Ta}}} \left(f_{\text{Ta}}(X_{\text{Ta}}^{(i)}; \theta_0, \theta_{\text{Ta}}) - L_{\text{Ta}}^{(i)} \right)^2. \tag{4.8}$$

8: **Output:** Final adapted LSTM metamodel $f_{\text{Ta}}(\cdot; \theta_0, \theta_{\text{Ta}})$ for the new VA contracts in the target task.

- the same contract but different underlying asset model, and
- different contracts but the same asset model.

Similar to Section 3.5, we use the standard nested simulation procedure to generate the dataset for the source and target tasks. Serveral VA datasets generated with the standard nested simulation procedures under geometric Brownian motion (GBM) and regime-switching GBM (RS-GBM) asset models. In this experiment, we use the same RS-GBM asset model as Chapter 3 to generate the dataset for the source and target tasks. When following a static lapse model and a dynamic lapse model, the monthly lapse rate is given by Equation 3.14 and Equation 3.13, respectively. The experiments are run on a machine with a AMD Ryzen 9 7900X processor with 32 GB of RAM and a Nvidia RTX 3060 Ti GPU

Contract	**Asset Model**	**Lapse**	$M_{\mathbf{So}}$	$M_{\mathbf{Ta}}$
GMMB	RS-GBM	No lapse	50,000	N/A
GMMB	RS-GBM	Static lapse	50,000	2000
GMMB	RS-GBM	Dynamic lapse	50,000	2000
GMWB	RS-GBM	Dynamic lapse	N/A	2000

Table 4.1: VA Contracts for Transfer Learning Experiments

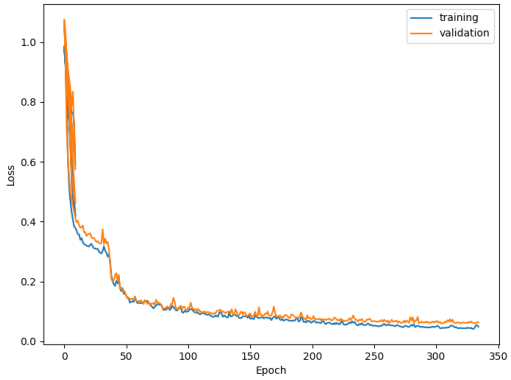
Table 4.1 lists the VA contracts used in the TL experiments. These contracts include GMMB and GMWB contracts with no lapse, static lapse, and dynamic lapse features. These contracts include GMMB and GMWB contracts with no lapse, static lapse, and dynamic lapse features. Before transferred to the target task, a LSTM metamodel is trained on a source dataset with $M_{\text{So}} = 50{,}000$ samples generated with $N_{\text{So}} = 100$ inner replications[1]. The pre-trained LSTM metamodel is then adapted to a target task that is different from the source task. The training dataset for the target task has $M_{\text{Ta}} = 2000$ samples generated with $N_{\text{Ta}} = 100$ inner replications. During the training on both the source and target tasks, 10% of the data is split into a validation dataset to monitor the training process and prevent overfitting by using early stopping. The complexity of the simulation schemes increases from the first task to the last task, and the LSTM metamodels need to adapt to the new conditions with a limited amount of training data on the target tasks.

[1]The GMMB contract on the GBM asset model is an exception, where scenario-wise contract losses can be computed analytically.

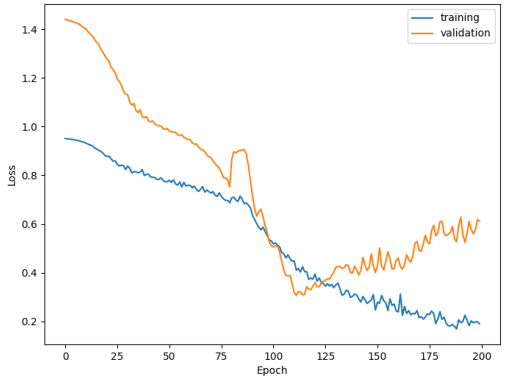
All LSTM metamodels are evaluated based on their training history graphs for the target task, which plot training and validation MSE against the number of training epochs. The training history graphs visualize the learning curves of the LSTM metamodels, which provide insights into stability and convergence behavior the metamodels. We measure the generalization performance of the LSTM metamodels using the true MSE, which quantifies the accuracy of the LSTM metamodels in approximating the true inner simulation model of VA contracts. The true MSE is computed by comparing the metamodel predictions with the true contract losses, which are approximated by the standard nested simulation procedure with 100,000 inner replications. Hyperparameters and network architectures for the LSTM metamodels are kept the same as the LSTM metamodel in Section 3.5. Due to a limited amount of computational resources, macro replications are not feasible for the TL experiments.

4.3.1 Learning Lapse Features

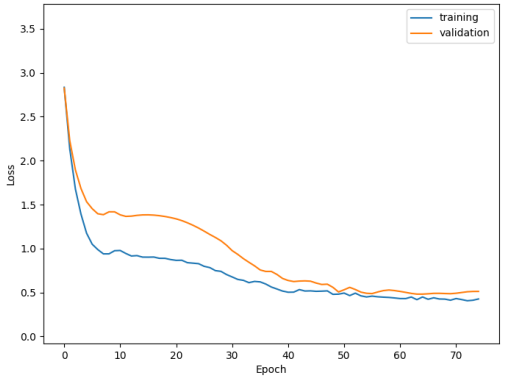
We first examine the performance of fine-tuning and layer freezing in adapting LSTM metamodels to new VA contracts with lapse features. Figure 4.1 compares the performance of LSTM metamodels on the target tasks with and without TL. Learning histories of the LSTM metamodels is shown for the target task of metamodeling GMMB contract losses on the RS-GBM asset model with static lapse. The source task is hedging a GMMB contract on RS-GBM with no lapse, and the target task is hedging the same GMMB contract but with a static lapse. The simulation data for the target task is generated with the same nested simulation procedure as the source task, except for the lapse feature. Figure 4.1a demonstrates the performance of the LSTM metamodel trained extensively only on the target task by using $M_{\text{TA}} = 50{,}000$ samples, which serves as a benchmark for this study. The metamodel achieves a low validation MSE due to the availability of a large dataset. In contrast, Figure 4.1b presents the results of training directly on the target task without pre-training on the source task. With only $M_{\text{TA}} = 2{,}000$ samples, the LSTM metamodel struggles to learn the temporal dependencies and patterns in the target task. It leads to highly unstable training dynamics with a substantial amount of fluctuations in the validation MSE. Learning without knowledge transfer leads poor generalization and extreme overfitting. Often, the training data is limited for a new VA contract, and such instability is particularly problematic for a quick adaptation of LSTM metamodels. TL techniques like fine-tuning and layer freezing can help mitigate these challenges. Figure 4.1c shows the results of fine-tuning a pre-trained metamodel on RS-GBM with no lapse. Fine-tuning offers a noticeable improvement over training without TL by reducing the instability in the validation MSE. However, despite this improvement, fine-



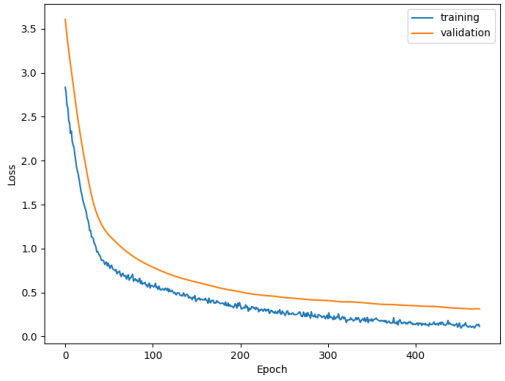
(a) Extensive Training on Target Task



(b) Without TL



(c) With Fine-tuning



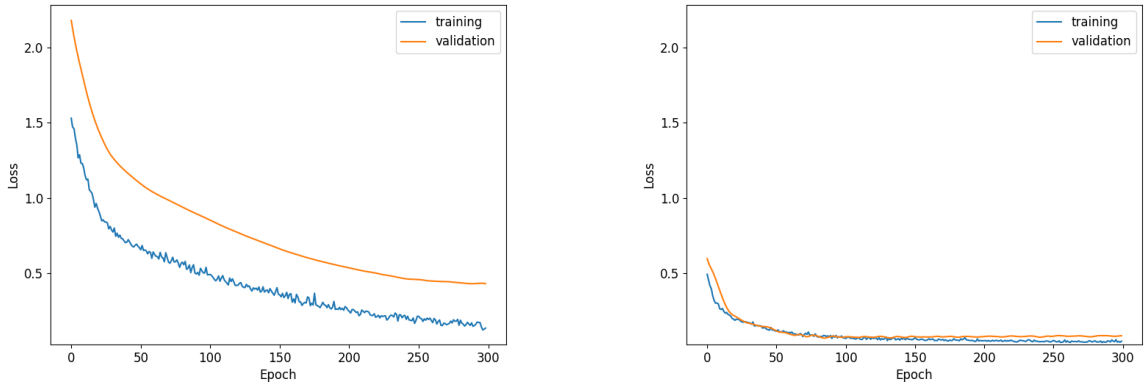
(d) With Layer Freezing

Figure 4.1: Metamodel performance on RS-GBM GMMB with static lapse

tuning may not fully mitigate the challenges of training with a small dataset. No negative trasfer is observed, but the fine-tuned metamodel struggles to achieve a low validation MSE. Lowering the learning rate does not help the fine-tuned metamodel converge, and the validation MSE remains high after increasing the number of training epochs. It indicates that fine-tuning may not be sufficient for a limited amount of training data. Figure 4.1d presents the performance of layer freezing on the same target task. Layer freezing offers a more stable training process compared to fine-tuning, with a lower validation error and reduced fluctuations during its training. The LSTM layers are critical for capturing general feature representations that are transferable across tasks, and freezing these layers helps

prevent overfitting to the target task. In addition, the layer freezing approach tunes fewer neural network parameters than crude fine-tuning. It allows the transferred metamodel to only focus on learning the lapse features without excessively adjusting the general temporal representations learned from the source task for the target task. This reduction in trainable parameters also accelerates the convergence, and it leads to a more stable and efficient training on the target task.

4.3.2 Transfer to VAs with a Dynamic Lapse



(a) Fine-tuning from No Lapse

(b) Fine-tuning from Static Lapse

Figure 4.2: Fine-tuned Metamodel performance on RS-GBM GMMB with a dynamic lapse

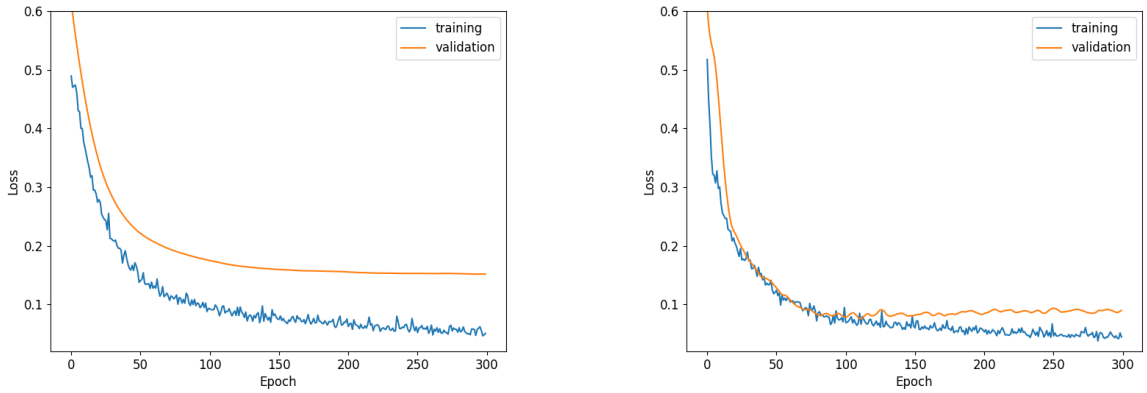
We further investigate the performance of fine-tuning on the target task of metamodeling GMMB contract losses on RS-GBM with a dynamic lapse. The source tasks used for pre-training are GMMB contracts on RS-GBM with no lapse and a static lapse, respectively. Figure 4.2 displays the learning curves of the LSTM metamodels fine-tuned from these two distinct source tasks. We observe that the performance of the fine-tuned metamodel is highly dependent on the similarity between the source and target tasks. Fine-tuning from a source task with a static lapse results in a faster convergence and a lower validation error. The metamodel trained on the GMMB with a static lapse captures some features that are beneficial for the GMMB with a dynamic lapse, which leads to a more stable training process. In Figure 4.2a, the metamodel needs to learn the effect of

- whether lapse is present, and

- whether the lapse is dynamic.

This makes learning more challenging.

The observed improvement in transferability when fine-tuning from a static lapse source task highlights the importance of selecting appropriate source tasks in TL. When the source task diverges substantially from the target task, the transferred metamodel may struggle to adapt to the new conditions given the limited amount of training data.



(a) Freezing LSTM Layers

(b) Freezing the Fully Connected Layer

Figure 4.3: Layer Freezing on RS-GBM GMMB with dynamic lapse

Figure 4.3 continues the investigation of layer freezing in the task of metamodeling GMMB contract losses on RS-GBM with the dynamic lapse. When transferring knowledge from the GMMB model with the static lapse, the primary adaptation for the target task involves learning the impact of the dynamic lapse. Freezing the LSTM layers and fine-tuning only the fully connected layer results in a higher validation error, which suggests a tendency of overfitting. This indicates that the fully connected layer struggles to adapt to the changes in the temporal dynamics introduced by a dynamic lapse, which can be viewed as another source of randomness in the time series. In contrast, freezing the fully connected layer and fine-tuning the LSTM layers leads to a lower validation error and better generalization. This can be attributed to the fact that the LSTM layers are responsible for capturing the temporal dependencies associated with the dynamic lapse, and the fully connected layer predicts the contract losses based on these learned features.

This experiment emphasizes the importance of choosing which layers to freeze based on the nature of the source and target tasks. In the case of learning the dynamic lapse

features, freezing the LSTM layers is not beneficial as they need to adapt to the new temporal patterns.

Lapse Type	Extensive	Fine-tuning	Layer Freezing	Without TL
No Lapse	N/A	0.4894	0.3361	N/A
Static Lapse	N/A	0.0794	0.0763	N/A
Dynamic Lapse	0.0587	N/A	N/A	0.2950

Table 4.2: Comparison of different TL methods on GMMB contracts

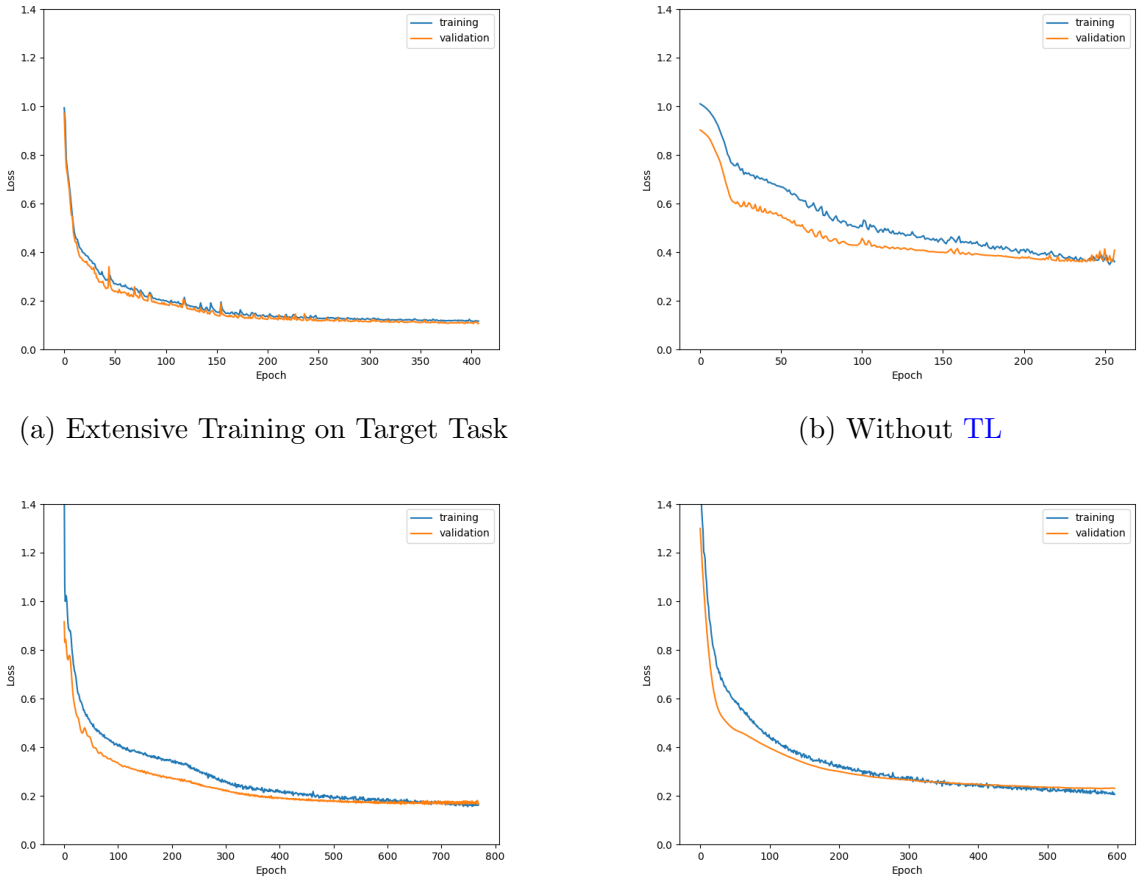
Table 4.2 summarizes the true MSEs of the LSTM metamodels trained using different TL methods across various source tasks. The calculations are based on the metamodel predictions and the true contract losses approximated with 100,000 inner replications. The first two rows show the performance of transferring knowledge from GMMB contracts with no lapse and the static lapse to the target task of GMMB with the dynamic lapse. The last row shows the performance of training without TL on the target task. The results demonstrate the effectiveness of fine-tuning and layer freezing in transferring knowledge from a related source tasks to the target task. When transferring from a source task with the static lapse to a target task with the dynamic lapse, the MSEs achieved are 0.0794 for fine-tuning and 0.0763 for layer freezing. The benchmark MSE of 0.0587, obtained from extensive training on the GMMB dynamic lapse task with much more samples.

While TL from static lapse GMMB do not reach this level of accuracy due to the limited data in the target task, they substantially outperform training without TL. This indicates that the models pre-trained on a static lapse setting are effective in capturing relevant features that are transferable to the dynamic lapse scenario.

However, when the source task is less similar to the target task, the benefits of TL are less pronounced. The MSEs in this case are higher, with fine-tuning resulting in 0.4894 and layer freezing achieving 0.3361. This suggests that the divergence between the source and target tasks can lead to negative transfer. These results highlight the importance of selecting source tasks that share significant similarities with the target task to maximize the effectiveness of TL. When the source and target contracts are substantially different, the pre-trained metamodels may struggle to adapt to the new conditions, as they may not have learned features relevant to the target task. There is no negative transfer observed, but the knowledge transfer is limited.

4.3.3 Transfer Knowledge to other Contract Types

When transferring knowledge from one VA contract type to another, the LSTM metamodel needs to adapt to different contract features and time series dynamics. We consider the task of metamodeling GMWB contract losses on RS-GBM asset model with the dynamic lapse, with the source tasks being a GMMB contract on RS-GBM also with the dynamic lapse. Figure 4.4 illustrates the learning history of transferring pre-trained LSTM metamodels from the GMMB contracts to the GMWB contracts.



(a) Extensive Training on Target Task

(b) Without TL

(c) With Fine-tuning

(d) With Layer Freezing

Figure 4.4: TL performance on RS-GBM GMWB with dynamic lapse

Figure 4.4a presents the performance of an extensively trained LSTM metamodel on

the target task using 50,000 samples, serving as a benchmark of the best metamodel performance. The extensively trained metamodel achieves a low validation error and demonstrates stable convergence due to having enough training samples. In contrast, Figure 4.4b shows the results of training the metamodel directly on the target task with 2000 training samples. Fewer training samples and no prior knowledge leads to unstable training dynamics. The validation MSE fluctuates substantially.

When applying fine-tuning from the GMMB source task (Figure 4.4c), the metamodel exhibits an improved performance compared to the training without TL. Fine-tuning results in a lower validation error and more stable convergence. Despite the differences between the GMMB and GMWB contracts, there is still valuable information that can be transferred. Both the LSTM layers and the fully connected layers capture general temporal patterns and feature representations that are beneficial for the target task. Fine-tuning allows the metamodel to adjust all its neural network layers. It allows a better adaptation to the complexities introduced by the GMWB contract.

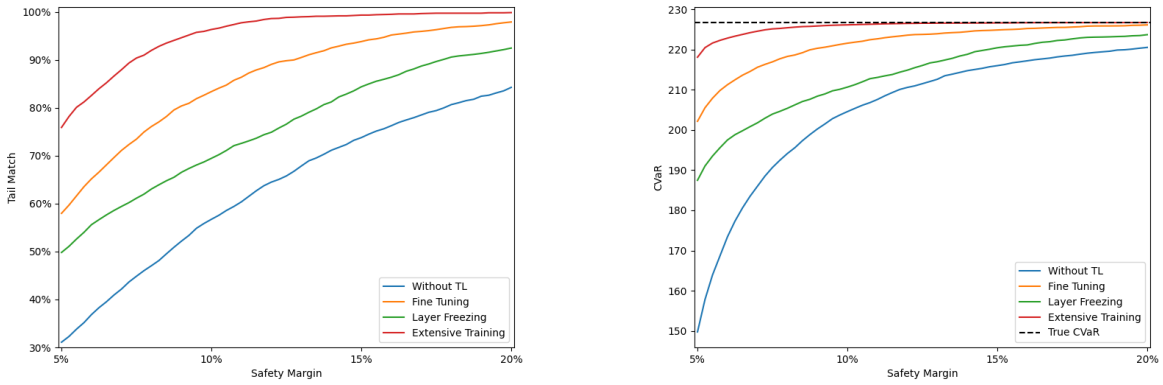
However, when employing layer freezing (Figure 4.4d), the metamodel's performance deteriorates. Negative transfer is observed. Freezing some layers trained on the GMMB contracts does not allow the metamodel to sufficiently adapt to the complexities of the GMWB contracts. The GMWB contracts are inherently more complex than GMMB contracts due to the guaranteed withdrawal benefits at each time step, and the contract features are substantially different. The complexity introduced by the GMWB contracts leads to significant changes in the time series dynamics that the metamodel needs to capture. Freezing the LSTM layers or the fully connected layer hinders the metamodel's ability to learn these new patterns, and it leads to a poor generalization and unstable error curves.

Model	**Training samples**	**Training MSE**	**True MSE**
Without TL	2000	0.3588	0.4188
Fine-tuning	2000	0.1690	0.1780
Layer freezing	2000	0.1828	0.2295
Extensive training	50,000	0.0853	0.0726

Table 4.3: Comparison of different TL methods to GMWB contracts

Table 4.3 summarizes the true MSEs of the LSTM metamodels on the GMWB contracts. Training MSE and true MSE are computed using the training labels and the true contract losses approximated with 100,000 inner replications, respectively. This is consistent with the terminology used in Chapter 3. The TL methods are compared to training from the scratch. The suboptimal performance of layer freezing in this context indicates

that the difference between the source and target tasks is too substantial for this method to be effective. While layer freezing can be advantageous when the source and target tasks are closely related, it may hinder performance when the tasks diverge substantially. In such circumstances, fine-tuning provides a better approach by allowing the metamodel to leverage transferable knowledge while adapting to the new task's specific requirements. Fine-tuning enables both the LSTM and the fully connected layers to update their weights, capturing the complex dynamics of the GMWB contracts more effectively. This is particularly beneficial when developing metamodels for the complex VA contracts with limited simulation data. For instance, transferring information from the GMMB contracts can still be valuable when modeling the GMWB contracts. Both contracts share some common contract features and temporal patterns, which allows a pre-trained LSTM metamodel to capture generalizable features that provide a solid foundation for the target task.



(a) Identifying Tail scenarios

(b) 95%-CVaR predictions

Figure 4.5: TL performance on RS-GBM GMWB with dynamic lapse

Figure 4.5 illustrates the performance of the two-stage procedure with TL for GMWB contracts with the dynamic lapse in predicting the tail scenarios and the 95%-CVaR. The graph provides a visual representation of how different TL approaches compare to training from the scratch and the standard nested simulation procedure. Fine-tuning consistently outperforms both layer freezing and training from the scratch across different simulation budgets, particularly in predicting tail scenarios and estimating the 95%-CVaR.

This finding is consistent with the training history and MSEs in Figure 4.4 and Table 4.3, respectively. It is important to note that while these results are promising, they also highlight the complexity of modeling the GMWB contracts with the dynamic lapse.

The fact that fine-tuning outperforms layer freezing suggests that there are substantial differences in the tail behavior of the GMMB and GMWB contracts, particularly when the dynamic lapse is considered. These findings highlight the need for careful model selection and validation when applying the TL techniques to different VA products. When the source and target tasks are substantially different, all layers need to be updated. Freezing some layers may lead to negative transfer.

4.3.4 Multi-task Learning

Multi-task learning enables the LSTM metamodels to learn shared representations across related VA contracts. In this section, we examine the performance of multi-task learning applied to two types of VAs, GMMB and GMWB with the dynamic lapse rates. The simulation datasets contain 2,000 samples for each contract type, and the LSTM metamodels are trained using multi-task learning. In our experiments, Algorithm 6 is used to train the LSTM metamodels simultaneously to minimize the multi-task MSE loss function in Equation (4.6). We use individual task training as a baseline for comparison, where the LSTM metamodels are trained separately on the GMMB and GMWB contracts. The objective is to assess how multi-task learning can improve the training efficiency and performance of both products compared to individual task training.

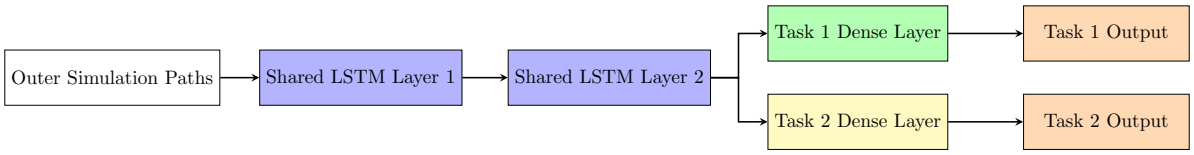
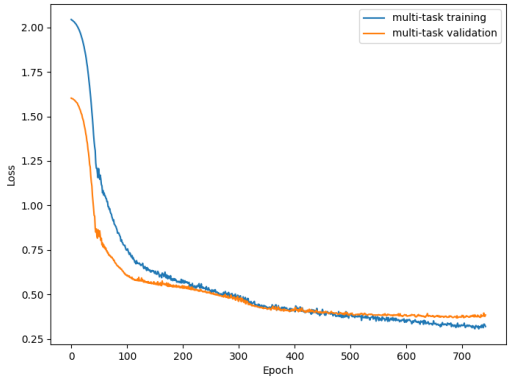


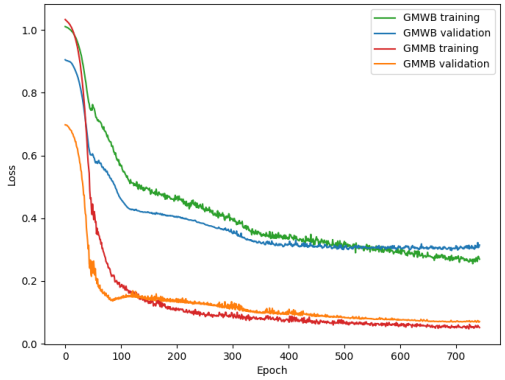
Figure 4.6: Multi-task learning framework for VA contracts

Figure 4.6 illustrates our multi-task learning framework for GMMB and GMWB contracts with the dynamic lapse rates. The outer simulation paths for both contracts are the same, which are generated from the same nested simulation procedure with 100 inner replications. The LSTM metamodels are trained simultaneously on both contracts. The LSTM layers are shared to train for capturing general temporal patterns, and each contract has its own fully connected layers that are trained for contract loss predictions.

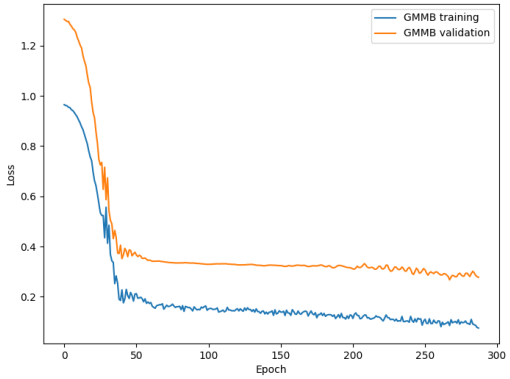
Figure 4.7 compares the learning curves for the training of the GMMB and the GMWB, with and without the multi-task learning. The comparison between the mult-itask learning (Figures 4.7a and 4.7b) and the individual task training ((Figures 4.7c and 4.7d)) demonstrates the benefit of the multi-task learning for both products.



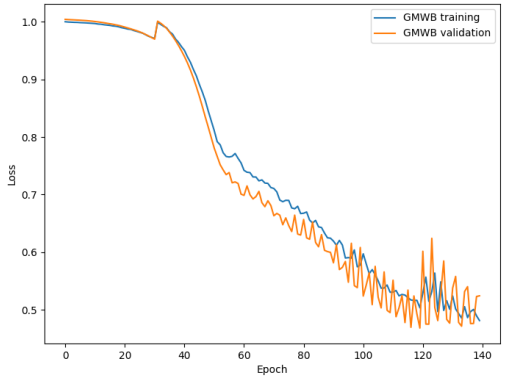
(a) Multi-task training history



(b) Task performance with multi-task training



(c) GMMB individual task training



(d) GMWB individual task training

Figure 4.7: Multi-task Learning on RS-GBM GMMB and GMWB with dynamic lapse

In the case of the GMMB, the multi-task learning allows the model to achieve faster convergence while reducing overfitting. For the GMWB, the multi-task learning framework helps to stablize the training process. The shared LSTM layers in the multi-task model are able to capture common temporal patterns and dynamics, which benefits both the GMMB and the GMWB when the training samples are scarce.

Similar to the pooling in Chapters 2 and 3 but on a higher level, multi-task learning enables the LSTM metamodels to leverage shared representations across related VA contracts.

4.4 Conclusion

In this chapter, we have introduced a TL framework to accelerate the training of LSTM metamodels for dynamic hedging of variable annuity contracts. Traditional nested simulation procedures for VA risk management are computationally intensive. LSTM metamodels offer a data-driven approach to approximate the true contract losses, which can substantially reduce the computational cost of hedging a single VA contract. However, training LSTM metamodels on new VA contracts can be challenging due to the limited availability of simulation data. Our proposed framework leverages pre-trained LSTM networks and TL techniques to adapt metamodels quickly to new but related VA contracts with a minimum amount of additional simulation cost. Fine-tuning a pre-trained LSTM metamodel on a new target task with a limited amount of data substantially improved training stability and predictive accuracy compared to training from the scratch.

Layer freezing further enhanced performance by retaining transferable temporal representations learned from the source task. However, the success of these methods depends on the similarity between the source and target tasks. When the tasks were closely related, freezing some neural network layers yields substantial benefits. Conversely, when the source and target tasks diverged substantially, fine-tuning the entire network is more effective.

Multi-task learning offers a robust framework for transferring knowledge across related simulation schemes in the LSTM metamodeling for nested simulations. By sharing representations through the LSTM layers, the model can learn more generalizable features that are beneficial across different VA contracts. The multi-task approach is particularly beneficial when training data for individual tasks is scarce, as it effectively pools information across tasks to enhance learning efficiency and predictive performance. We have extended the idea of pooling in Chapter 2 and 3 to a higher level. Furthermore, a metamodel trained with multi-task learning can serve as a pre-trained model. It is adaptable to various VA contracts and asset models, and it is a versatile tool for practical applications in dynamic hedging of VAs.

The integration of TL in LSTM metamodeling represents an important advancement in robust risk management associated with VA contracts. By effectively leveraging existing knowledge, financial institutions can maintain accurate and responsive risk management practices in a rapidly changing market environment. The methodologies presented in this chapter attempt to make a contribution to the broader applications of TL in financial modeling and risk assessment.

Chapter 5

Conclusion

In this thesis, we have explored the application of nested simulation procedures in the risk management of financial derivatives and insurance products, with a particular focus on variable annuity contracts featuring GMMB and GMWB. Recognizing the computational challenges inherent in the standard nested simulation procedure, especially when estimating tail risk measures for complex financial products, we have proposed an innovative approach that integrates machine learning-based metamodels into the nested simulation framework.

We have shown that the nested simulation framework can be used to efficiently estimate risk measures of complex financial products, such as VAs, by combining machine learning-based metamodels with nested simulations.

Our primary contribution lies in the development and implementation of LSTMs as metamodels for the inner simulations within the nested simulation procedure. By employing LSTMs, we have effectively approximated the contract losses associated with dynamic hedging strategies under various market scenarios. This approach has substantially reduced the computational complexity of the standard nested simulation procedure while maintaining high levels of accuracy in risk estimation. The LSTM metamodels have captured the temporal dependencies and nonlinear relationships present in financial time series data, enabling efficient estimation of tail risk measures.

Through extensive numerical experiments and sensitivity analysis, we have demonstrated that our LSTM-based nested simulation procedure provides accurate and computationally efficient estimates of the tail risk associated with VA contracts. The results have shown that the metamodels can effectively replace the computationally intensive inner simulations in the standard nested simulation procedure. This approachhas reduced the overall runtime without compromising the precision of risk assessments. This efficiency

gain is particularly valuable for insurers and financial institutions that require timely and accurate risk evaluations to meet regulatory requirements and make informed and timely risk management decisions.

Additionally, we have investigated the feasibility of our approach under changing market conditions and different VA contract features. By incorporating TL techniques, we have enabled rapid adaptation of the LSTM metamodels to new contracts and evolving market dynamics. This aspect of our research highlights the potential for machine learning models to not only improve computational efficiency but also enhance the flexibility and responsiveness of risk management practices in the face of market volatility.

This thesis is an attempt to bridge the gap between traditional simulation schemes and modern machine learning techniques in the context of financial risk management. By combining advanced machine learning techniques with financial risk management practices, this thesis contributes to the development of more efficient, adaptable, and robust methodologies that are essential in the rapidly evolving landscape of quantitative finance and actuarial science.

Chapter 6

Future Work: Deep Hedging Variable Annuities with Transfer Learning

In the evolving landscape of financial markets, insurance products such as VAs have gained a substantial amount of interest due to their ability to provide both investment growth and guaranteed benefits. Managing the risks associated with these products, especially in volatile market conditions, is a complex task that demands sophisticated financial modeling techniques. Reinforcement Learning (RL) has emerged as a powerful tool for addressing such complexities. With the recent advancement in deep learning, DNN architectures have been successfully applied to RL problems, leading to the development of deep reinforcement learning (Deep Reinforcement Learning (DRL)) algorithms. These algorithms have demonstrated remarkable success decision-making tasks that must be optimized over time. In the context of managing VAs, DRL has shown promise in optimizing hedging strategies where long-term financial and policyholder decisions interact dynamically with changing market conditions.

Reinforcement learning is a branch of machine learning in which an agent learns to make sequential decisions by interacting with its environment. Through trial and error, the agent receives feedback in the form of rewards or penalties, allowing it to learn an optimal policy for achieving long-term goals. In financial and actuarial contexts, RL has been applied to portfolio optimization, asset allocation, and risk management. In a dynamic hedging problem, the agent is referred to as the decision-maker that tries to minimize financial risks over time horizon by adjusting the hedging portfolio based on the current environment, which is the current state of the financial markets and actuarial assumptions. By integrating deep neural network, DRL extends this framework by enabling the agent to handle high-dimensional and complex environments. DRL has shown particular promise in

finance because of its ability to process vast amounts of financial data and discover intricate patterns in market dynamics. Despite their success, DRL faces practical limitations when applied to real-world financial markets. Its most significant drawback is the requirement for extensive data and computational resources. Training a DRL agent from the scratch requires a large dataset along with considerable computational time for the purpose of exploration and learning. This is especially problematic for VA contracts, which are long-term insurance products that are sensitive to market fluctuations and underlying actuarial assumptions. Given that financial markets are highly dynamic, a frequent retraining to account for new market regimes becomes necessary, which can lead to prohibitively high costs and computational inefficiency. Moreover, using historical data leads to overfitting to the past. An overfitting model can lead to suboptimal hedging strategies that perform poorly in changing market conditions.

TL offers a compelling solution to the challenges posed by existing DRL approaches. Similar to the TL framework proposed in Chapter 4, TL can also be applied to DRL to accelerate the training. In the context of financial modeling, instead of training an RL agent from the scratch every time underlying assumptions change, the model can utilize the knowledge gained from past market environments to accelerate the learning process under the new environment. TL in financial markets is particularly advantageous for addressing the data scarcity problem. By transferring existing knowledge from similar environments, TL improves model performance in situations where historical data is limited or noisy. This allows models using TL to be both more accurate and computationally efficient. Additionally, TL enhances the robustness of models by allowing continuous learning. Instead of retraining a deep RL model from the scratch when market conditions shift, TL enables the reuse of pre-trained models, facilitating a faster adaptation to new environments while avoiding overfitting to past conditions. In the risk management of VA contracts, transfer learning enables quicker adaptation to changing financial and actuarial assumptions by reusing information obtained from another set of related assumptions, such as the management of similar financial instruments or different market regimes. A DRL agent equipped with transfer learning can adjust its strategies in response to new market data, making the model more adaptive and responsive without the need for costly retraining. More specifically, a DRL agent trained to manage VA contracts for a GMMB contract can apply its learned strategies to a GMWB contract with the same financial assumptions. A DRL agent trained for a VA product on a Black-Scholes asset model can transfer its knowledge to the training of a similar product on a stochastic volatility model. This approach substantially reduces the computational costs associated with the training of new VA products with different assumptions and improves the generalization of the model to unseen market conditions. By integrating the transfer learning with the existing DRL frameworks, we can

eliminate the inefficiencies of training models from scratch, provide faster adaptability to new market conditions, and enhance the scalability and computational efficiency of financial models. This approach offers a more robust and practical solution for managing VA contracts in dynamic and volatile financial markets.

The rest of the chapter is organized as follows: Section 6.1 introduces the mathematical formulation for hedging VAs and defines the key components of the framework. Section 6.2 provides an overview of existing RL algorithms and their limitations in the context of hedging VAs. Section 6.3 introduces the proximal policy optimization algorithm and its application to the hedging of VAs. Finally, Section 6.6 concludes the chapter and discusses future research directions.

6.1 Markov Decision Process for Hedging VAs

A Markov Decision Process (MDP) provides a mathematical framework for modeling decision-making in stochastic environments over time. By formulating the VA hedging problem as an MDP, we can utilize RLalgorithms to learn optimal hedging policies directly from data. An MDP is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is a state transition probability distribution, where $\mathcal{P}(s_t|s_{t-1}, a_{t-1})$ represents the probability of transitioning from state s_t to state s_{t-1} due to action a_{t-1}.
- $\mathcal{R}$ is a reward distribution, $\mathcal{R}(s, a, s')$ is the reward received after transitioning from state s to state s' due to action a.
- γ is a discount factor, $\gamma \in [0, 1]$, which models the present value of future rewards.

The objective in an MDP is to find a policy, $\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A})$, that maximizes the cumulative reward over time. A MDP provides a mathematical framework for solving sequential decision-making tasks in an enviornment where outcomes are partly random and partly under the control of a hedging agent. For hedging VAs, the asset dynamics and the mortality model are not controlled by the agent, but the agent controls the cumulative reward by deciding on the hedging weights. In this section, we reformulate the hedging enviornment of VAs from Section 3.2.2 as an MDP and define the key components of the MDP.

6.1.1 Hedging Environment of Variable Annuities

Consider a generic VA contract with maturity $T > 0$ periods, e.g., $T = 240$ months. Then the contract expires at $T' = \min\{T, \tau\}$, i.e., the earlier of the contract maturity and the death of the policyholder. Let S_t, F_t, and G_t be the indexed stock price, the subaccount value and the guarantee value, respectively, at time $t = 1, 2, \ldots, T$. Evolution of the subaccount value and the guarantee value of a VA contract affect the contract payout. For clarity, we use F_t and F_{t_+} to denote the sub-account value just before and just after the withdrawal at time t, if any. Let η_g be the gross rate of management fee that is deducted from the fund value at each period and let $\eta_n < \eta_g$ be the net rate of management fee income to the insurer. The difference between the gross management fee and the net management fee income represents the incurred investment expenses.

At the inception of the contract, i.e., $t = 0$, we assume that the whole premium is invested in the stock index and the guarantee base is set to the sub-account value:

$$S_0 = F_0 = G_0.$$

At each time $t = 1, \ldots, T$, the following events take place in the following order:

1. The dynamic lapse rate q_t is applied to the contract, i.e., q of the policyholders leave the contract at time t.

$$q_t = q_t^B \cdot \text{clip}(M_q(\frac{G_{t-1}}{F_{t-1}} - D_q), L_q, U_q), \tag{6.1}$$

   where q_t^B is the base lapse rate, $\text{clip}(x, a, b)$ is a function that clips the value of x to be within the range $[a, b]$, and M_q, D_q, L_q, and U_q are the model parameters taken from National Association of Insurance Commissioners' (NAIC) Valuation Manual 21 (NAIC, 2021).

2. The sub-account value changes according to the growth of the underlying stock and the (gross) management fee is deducted. That is,

$$F_t = F_{(t-1)_+} \cdot \frac{S_t}{S_{t-1}} \cdot (1 - \eta_g) \cdot (1 - q_t), \tag{6.2}$$

   where $(x)^+ = \max\{x, 0\}$ and $F_{(t-1)_+}$ will be defined later. The insurer's income at time t is the net management fee, i.e., $F_t \eta_n$.

3. The guarantee value ratchets up (ratcheting is a common feature in GMWB) if the sub-account value exceeds the previous guarantee value, i.e.,

$$G_t = \max\{G_{t-1} \cdot (1 - q_t), F_t\}. \tag{6.3}$$

A GMMB can be modeled with $G_t = G_{t-1} \cdot (1 - q_t)$.

4. The withdrawal is made (for GMWB) and is deducted from the sub-account value, i.e.,

$$F_{t_+} = (F_t - I_t)^+, \tag{6.4}$$

where $I_t = \xi G_t$. A GMMB can be modeled with $\xi = 0$.

Consider a VA contract whose delta hedge portfolio at any time t, $t = 0, 1, \ldots, T-1$, consists of Δ_t units in the underlying stock and B_t amount of a risk-free zero-coupon bond maturing at time T. The value of the hedge portfolio at time $(t-1)$ is:

$$H_{t-1} = \Delta_{t-1} S_{t-1} + B_{t-1},$$

where S_t is the underlying stock price and any time $t > 0$. This hedge portfolio is brought forward to the next rebalancing time t, when its value becomes:

$$H_t^{bf} = \Delta_{t-1} S_t + B_{t-1} e^{-rt}.$$

Therefore, the time t hedging error, i.e., the cash flow incurred by the insurer due to rebalancing at time t, is

$$HE_t = H_t - H_t^{bf}, \quad t = 1, \ldots, T-1. \tag{6.5}$$

The P&L of the VA contract includes the cost of the initial hedge (H_0), the hedging errors (3.8), the unwinding of the hedge at maturity (H_T^{bf}), and the contract payout at time $t \in \{0, \ldots, T\}$. Mathematically, the present value of the liability for a GMMB contract is

$$\begin{aligned} L &= H_0 - e^{-rT} H_T^{bf} + \sum_{t=1}^{T-1} e^{-rt} HE_t + \sum_{t=1}^{T} e^{-rt} C_t - \sum_{t=1}^{T} e^{-rt} F_t \eta_n + e^{-rT} (G_t - F_t)^+ \\ &= e^{-rT} (G_T - F_{T^+})^+ + \sum_{t=1}^{T} e^{-rt} \left(\Delta_{t-1} (S_{t-1} - e^{-r} S_t) + C_t - F_t \eta_n \right) \end{aligned} \tag{6.6}$$

where the equality holds by a rearrangement of terms and a telescopic sum simplification of $e^{-rt} B_t$, $t = 0, \ldots, T-1$, and $C_t := C \cdot S_{t-1} \cdot |\Delta_t - \Delta_{t-1}|$ is the tranaction cost at time

t (Gârleanu and Pedersen, 2013). Hence, the present liability consist of the hedging errors, the transaction costs, the net management fees, and the contract payouts. Similarly, the present value of the liability for a GMWB contract is

$$L = \sum_{t=1}^{T} e^{-rt} \left(\Delta_{t-1}(S_{t-1} - e^{-r} S_t) + C_t - F_t \eta_n \right) \tag{6.7}$$

The GMWB payout is implicitly included in $F_t \eta_n$. Both Equation (6.6) and Equation (6.7) neglect the transaction cost to liquidate the hedge portfolio at maturity. The discrete-time hedging problem of VAs can be conveniently formulated as an MDP, where the states, actions, transition probabilities, policies, and rewards are defined as follows. Different from the formulation in Chapter 3, the hedging weight $\Delta_0, \ldots, \Delta_{T-1}$ are not estimated from inner simulation steps but are directly controlled by a DRL agent. Thus, the hedging weight Δ_t at time t is estimated using a neural network whose input is the state $\mathbf{s}_t$ and output is Δ_t.

6.1.2 State Space

The state space $\mathcal{S}$ represents all the information needed to characterize the hedging environment. In a VA hedging problem, $\mathbf{s}_t$, the information available to a hedging agent at each time t is

$$\mathbf{s}_t := (S_t, F_t, G_t, \tau, \Delta_{t-1}) \in \mathcal{S}.$$

where τ is the remaining time to maturity and Δ_{t-1} is the previous hedging weight. The hedging environment is partially observed, e.g., the agent does not have access to the contract specifications. It has to learn a representation of the relevant information from the observed state $\mathbf{s}_t$.

6.1.3 Action Space

The action space $\mathcal{A}$ represents the set of actions that the agent can take at each time t. In a VA hedging problem, the action space only includes the hedging weight Δ_t.

6.1.4 Policy

A policy π determines the best action to take in a given state, i.e.,

$$a_t \sim \pi(\cdot|\mathbf{s}_t).$$

In a VA hedging problem the policy outputs the hedging weight $a_t = \Delta_t$ given the observed state $\mathbf{s}_t$. In the following sections, we will refer to Δ_t as the policy.

6.1.5 Transition Probabilities

The transition probabilities $\mathcal{P}$ represent the probability of transitioning from one state to another from taking an action. In our setup, the transition probabilities are fully determined by Equation (6.1), Equation (6.2), Equation (6.3), and Equation (6.4). Let $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t)$ be the probability of transitioning from state $\mathbf{s}_t$ to state $\mathbf{s}_{t+1}$ due to action a_t. Aside from the previous hedging weight Δ_{t-1}, the transition probabilities are independent from the actions. Paths simulated from the transition probabilities are called trajectories (or episodes) of the MDP.

6.1.6 Reward and Discount Factor

The reward function $\mathcal{R}$ and discount factor γ in a typical RL problem are defined as follows:

$$\begin{aligned} \mathcal{R} &= \sum_{t=1}^{T-1} \gamma^t R_t \\ \gamma &= e^{-r}, \end{aligned} \tag{6.8}$$

where R_t is the term reward received by the agent at time t, and the discount factor γ is conveniently inherited from the risk-free rate r. In hedging, the term reward R_t should be derived from Equation (6.6) and Equation (6.7) to represent the performance of the hedging policy, i.e., how closely the hedging portfolio tracks the liability. For the GMWB, we propose a term reward formulation that encourages the agent to maximize the expected reward with low hedging errors and transaction costs:

$$R_t = F_{t-1}\eta_n - \Delta_{t-1}|S_{t-1} - e^{-r}S_t| - C_{t-1}(\Delta_{t-1}), \quad t \in \{1, \dots, T\} \tag{6.9}$$

The reward function of the GMWB directly links to the time-t loss of a GMWB contract as in Equation (6.7). The GMMB loss consists of an additional payoff at maturity T, thus its term reward is formulated as:

$$R_t = \begin{cases} F_{t-1}\eta_n - \Delta_{t-1}|S_{t-1} - e^{-r}S_t| - C_{t-1}(\Delta_{t-1}), & t \in \{1, \dots, T-1\} \\ F_{t-1}\eta_n - \Delta_{t-1}|S_{t-1} - e^{-r}S_t| - C_{t-1}(\Delta_{t-1}) + (G_T - F_{T^+})^+ & t = T \end{cases} \tag{6.10}$$

6.1.7 Value Function and Advantage Function

This section introduces some critical concepts in RL, i.e., the value function and the advantage function. They are essential for understanding the performance of the hedging policy and the learning process of the RL agent.

A value function quantifies the expected cumulative reward over time. The state value function, $V^\pi(s_t)$, is defined as the expected cumulative reward starting from state s_t and following policy π thereafter:

$$V^\pi(s_t) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t | \pi, s_t\right], \tag{6.11}$$

where R_t is the reward received by the agent after taking action a_t in state s_t and transitioning to state s_{t+1}, and the expectation is taken over the possible sequences of states and rewards that follow from the policy π. The state-action value function, $Q^\pi(s_t, a_t)$, is defined as the expected cumulative reward starting from state s_t, taking action a_t, and following policy π thereafter:

$$Q^\pi(s_t, a_t) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t | \pi, s_t, a_t\right], \tag{6.12}$$

Based on Equations 6.11 and 6.12, the advantage function, $A^\pi(s_t, a_t)$, is defined as the difference between the state-action value function and the state value function:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t). \tag{6.13}$$

The advantage function quantifies the advantage of taking action a_t in state s_t over following the policy π.

6.2 Exising RL Algorithms and Their Limitations

6.2.1 Model-Based RL: From AlphaZero to Deep Hedging

Deep reinforcement learning first attacted attentions when AlphaGo (Silver et al., 2016) surpasses human in playing Go. Soon after, AlphaZero (Silver et al., 2016) emerged as a

model-based RL algorithm that improves over self-plays. It quickly extended beyond Go to other games like Chess and Shogi. In essence, the AlphaZero agent use a neural network to paramaterize the policy and the value function. The neural network takes the current state of the game as input and outputs the probability distribution of the next move and a value of the current state.

$$(\pi, v) = f_\theta(s), \quad L(\theta) = -\pi^\intercal \log(\mathbf{p}) + (z - v)^2 + c||\theta||^2, \tag{6.14}$$

where f_θ is a neural network with parameters θ, s is a current state of the game, π is a policy, v is a value of the current state, and c is a regularization parameter. $\mathbf{p}$ and z are a probability distribution of the next move and an outcome of the game, respectively. They are induced by a planning algorithm, i.e., Monte Carlo Tree Search (MCTS). Since it is infeasible to efficiently simulate trajectories in the game of Go, a planning algorithm is necessary to generate expected outcomes. The neural network f_θ is trained with a stochastic gradient descent algorithm to minimize L. L has three components. The last term penalizes large neural network weights. Its first two terms measure the similarity between $f_\theta(s)$, the learned policy and value function, with the action and game outcome planned by the MCTS. The neural network is trained to fit the planned outcomes of the game. It stores a global solution that predicts the outcome of the game and the next move before they are actually observed. This is known as model-based RL (Sutton and Barto, 2018). Using a model allows for planning, i.e., decision-making take place using future situations that have not been experienced by the agent.

In a typical hedging problem, the transition probabilities are independent from the hedging weights, i.e., the actions. This simplifies the planning as complete trajectories can be simulated without the need of a planning algorithm. In contrast to AlphaZero, deep hedging (Buehler et al., 2019) uses monte carlo simulations to generate sample paths of asset dynamics. These simulations are conducted based on predefined stochastic asset models. The parameters and structure of these models are specified in advance, dictating the behavior of the simulated asset prices.

A common choice of ρ is a CVaR. In practice, the neural network f_θ is trained to minimize the empirical CVaR over a batch of trajectories. In other words, planning is done through ρ, which measures the performance of a hedging policy over a batch of complete trajectories. The neural network, representing the hedging policy, is trained using the data generated from these simulations. The objective is to learn a strategy that minimizes a tail risk measure of the total present liability. Consequently, the learned policy is inherently tailored to the characteristics of the model used in the simulations. The ability to know the future outcomes of the hedging policy before they are actually observed and the use of

a risk measure to guide the training of the neural network are the key features of model-based RL. These models describe how asset prices evolve over time, and more importantly, they are used to predict future states based on current information. While model-based approaches can be powerful when the model accurately reflects reality, they are susceptible to model risk, i.e., the risk that the chosen model is incorrect or incomplete. It can lead to suboptimal hedging strategies if the real market deviates from the model assumptions.

For hedging VAs, Xu (2020) propose a deep hedging algorithm that falls into this category of model-based RL. A cost function is defined to minimize the expected shortfall of the total present liability. Carbonneau (2021) extends the model-based approach to hedging long-term financial derivatives and compares the performance of deep hedging strategies trained with different cost functions. In deep hedging (Buehler et al., 2019) the state space of the MDP contains the previous hedging weight. A more recent paper by Imaki et al. (2021) proposes a new network architecture that removes the previous hedging weight from the state space by introducing a no-transaction band. It allows for a more efficient training of the neural network by removing the semi-recurrent problem structure.

Despite the success of model-based RL in hedging, most studies focus on hedging different types of contracts on classical asset models, e.g., Black-Scholes model and Heston model. Motivated by the seminal work by Gatheral et al. (2022), there are a many subsequent works focusing on roughening the classical stochastic volatility models. In the rough volatility models, the latent volatility process is modelled by a stochastic process that is rougher than the sample paths of diffusion process. Among these rough volatility models, notable examples are rough Heston model and the rough Bergomi model. Model-based RL struggles in a rough volatility model due to difficulties in designing a suitable cost function. Horvath et al. (2021) shows that a standard implementation of deep hedging can lead to suboptimal hedging performance in a discretized rough Bergomi model.

In contrast, a model-free approach does not require a explicit design for different asset dynamics, making it more flexible and adaptable to different market conditions and more robust to model misspecification.

6.2.2 Model-Free RL for Hedging

In a model-free reinforcement learning approach, the agent is a trial-and-error learner that interacts with the environment to learn the optimal policy by iteratively updating the policy and the value function based on the observed states, actions, and rewards. For hedging VAs, the agent does not have access to a planning algorithm or the underlying

stochastic asset model, and it must learn the optimal policy by directly interacting with the environment.

Model-free RL algorithms gain its popularity in robotics tasks, where transition probabilities in the environment are unknown and can be affected by the agent's actions. In the context of hedging, existing studies use transaction costs as a function of the hedging weights, which makes the transition probabilities independent on the actions. However, the trade size can affect the asset price in a real environment (Hasbrouck, 1991), which makes the model-free approach a promising direction for hedging VAs in the real market.

Boardly speaking, model-free RL can be categorized into two types: value-based RL and policy-based RL. Value-based approaches learn the value function directly from the data and derive the policy from the learned value function. The value function is learned by training a DNN to approximate the value function, which quantifies the expected cumulative reward over time. The policy is then derived from the learned value function by selecting the action that maximizes the value function at each state. Mnih et al. (2015) is the first paper to demonstrate the effectiveness of value-based deep reinforcement learning in training agents to play Atari games. To update the value function, the DNN is trained to minimize the difference between the current value function and the target value function, which is updated based on the maximum expected reward from the previous iteration. When trained to approximate the value function, the DNN benefits from the use of experience replay buffers, which is proposed by Lin (1992) to store and sample historical transitions and uniformly sampled during training to enhance the sample efficiency and stability of the learning process. Kolm and Ritter (2019) uses a value-based deep reinforcement learning approach to hedge European options under the GBM asset model.

In valued-based approaches, the value function is updated iteratively with the maximum expected reward of all possible actions, which can be computationally expensive for continuous action spaces. To overcome this difficulty, a policy-based approach estimates the value function before realization of the final payoff of the hedging portfolio, which is particularly useful for hedging long-term financial derivatives or VA contracts. Two types of policy-based approaches prevail in the literature: actor-critic and policy gradient methods.

An actor-critic method designs an actor network and a critic network to learn the value function and the policy simultaneously. The actor network learns the policy by directly outputting the action based on the current state, while the critic network learns the value of an action by approximating the expected cumulative reward over time following the actor's policy. A popular actor-critic algorithm is the Deep Deterministic Policy Gradient (DDPG) algorithm proposed by Lillicrap et al. (2015), which is implemented by Xu and

Dai (2022) in hedging financial derivatives in S&P 500 and DJIA index options.

A policy gradient method uses policy gradient theorem to update the policy by performing gradient ascent with respect to the policy parameters.

Algorithm 8 Vanilla Policy Gradient (REINFORCE)

1: **Initialize** policy parameters θ arbitrarily.
2: **For** iteration $k = 0, 1, 2, \ldots$ **do**
3: Collect a set of $\mathcal{D}_k$ by sampling from the enviornment with policy π_θ.
4: Compute the rewards R_t for each trajectory in $\mathcal{D}_k$ for $t = 0, 1, \ldots, T$.
5: Update the policy parameters θ_{k+1} by

$$\theta_{k+1} = \theta_k + \alpha \frac{1}{|\mathcal{D}_k|T} \sum_{\kappa \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R_t$$

6: **End For**

Algorithm 8 shows the REINFORCE algorithm (Sutton et al., 1999), a classic policy gradient method that updates the policy by performing gradient ascent with respect to the policy parameters. When training on real-world data, deep hedging of Buehler et al. (2019) effectively becomes a model-free policy gradient methods. The policy is updated by performing gradient ascent with respect to the policy parameters to maximize the expected reward over time, where the reward function is designed to be the negative of a tail risk measure of the total present liability.

Representative algorithms include the Trust Region Policy Optimization (TRPO) algorithm proposed by Schulman et al. (2015a) and the PPO algorithm proposed by Schulman et al. (2017). Du et al. (2020) show that the PPO algorithm is more sample efficient and stable than the value-based approaches in training agents to hedge financial derivatives. The most relevant paper to our study is Chong et al. (2023), which uses the PPO algorithm to hedge GMMB with a GMDB rider under the Black-Scholes framework. However, information of the underlying risk factors is leaked to the agent, which makes their study pseudo-model-free. To further improve the risk management of VAs under rough volatility models, we propose a model-free deep reinforcement learning approach to hedge VAs without explicit or implicit knowledge of the underlying risk factors.

6.3 Hedging VAs with a PPO Agent

In a hedging problem, an insurer aims at finding a policy π that maximizes the expected reward over time. In a policy-based RL approach like the PPO, both the policy and the value function are parameterized by deep neural networks. We denote the policy by π_θ and the value function by V_ϕ, where θ and ϕ are the parameters of the value network and the policy network, respectively. The policy is updated according to an objective function that reflects the advantage of taking action a_t in state s_t over following the policy π.

$$\max_\theta \mathbb{E}\left[\min\left\{\Lambda_t(\theta)\hat{A}^{\pi_\theta}(s_t,a_t), \text{clip}(\Lambda_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_\theta}(s_t,a_t)\right\}\right] \tag{6.15}$$

$$\Lambda_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{6.16}$$

where θ_{old} denotes the policy parameters at the previous iteration, ϵ is a clip range hyperparameter that controls the size of the policy update, $\text{clip}(x,a,b)$ is a function that clips the value of x to be within the range $[a,b]$, and $\hat{A}^{\pi_\theta}(s_t,a_t)$ is an estimated advantage of taking action a_t in state s_t over following the policy $\pi_{\theta_{old}}$. In our implementation, the advantage function is estimated following the GAE method proposed by Schulman et al. (2015b). The policy is updated by performing gradient ascent with respect to the policy parameters θ to maximize the objective function, which is estimated by sampling from the environment and computing the empirical average of the objective function over the sampled trajectories. The first component in the objective function, $\Lambda_t(\theta)\hat{A}^{\pi_\theta}(s_t,a_t)$, is a surrogate objective function that approximates the advantage of taking action a_t in state s_t over following the policy $\pi_{\theta_{old}}$, and the second component, $\text{clip}(\Lambda_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_\theta}(s_t,a_t)$, is a clipped surrogate objective function that prevents large policy updates and ensures more stable training. The TRPO algorithm proposed by Schulman et al. (2015a) uses a similar clipped surrogate objective function to ensure that the policy update does not deviate too far from the old policy, but PPO simplifies the optimization problem by using a clipped surrogate objective function instead of a hard-coded constraint on the policy update. Compared to TRPO, PPO is more computationally efficient and easier to implement, making it a popular choice for training deep reinforcement learning agents. For a in-depth discussion on the implementation details of both TRPO and PPO, we refer the reader to Engstrom et al. (2020).

Algorithm 9 outlines the PPO algorithm for training an agent to hedge VAs. At each iteration k, the agent collects a set of trajectories $\mathcal{D}_k$ by interacting with the environment (Section 6.1.1) with the current policy π_{θ_k}. The agent then computes the rewards $\hat{R}_t$ for each trajectory in $\mathcal{D}_k$ and estimates the advantage function with $\hat{A}_{\pi_{\theta_k}}(s_t,a_t)$ using the value

Algorithm 9 PPO for Hedging Variable Annuities

1: Initialize policy parameters θ_0 and value function parameters ϕ_0.
2: **For** iteration $k = 0, 1, 2, \ldots$ **do**
3: Collect a set of trajectories $\mathcal{D}_k$ by sampling from the environment with policy π_{θ_k}.
4: Compute the rewards $\hat{R}_t$ for each trajectory in $\mathcal{D}_k$.
5: Estimates the advantage function with $\hat{A}_{\pi_{\theta_k}}(\mathbf{s}_t, a_t)$ using value function V_{ϕ_k}.
6: Update the policy parameters θ_{k+1} by

$$\max_{\theta} \mathbb{E}\left[\min\left\{\Lambda_t(\theta)\hat{A}^{\pi_\theta}(\mathbf{s}_t, a_t), \text{clip}(\Lambda_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_\theta}(\mathbf{s}_t, a_t)\right\}\right]$$

7: Update the value function parameters ϕ_{k+1} by minimizing the mean squared error between the predicted value and the actual collected reward.

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\kappa \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(\mathbf{s}_t) - \hat{R}_t\right)^2$$

8: **End For**

function V_{ϕ_k}. The policy parameters θ_{k+1} are updated by performing gradient ascent with respect to the policy parameters to maximize the clipped surrogate objective function. The value function parameters ϕ_{k+1} are updated by minimizing the mean squared error between the predicted value and the actual collected reward. The agent iterates this process until convergence to learn an optimal policy for hedging VAs.

6.3.1 Accounting for Non-Markovian Dynamics

In the context of hedging VAs, the state space is partially observed. The agent does not have access to the contract specifications and the underlying risk factors. Furthermore, a rough volatility model introduces non-Markovian dynamics, i.e., the future state of the system depends on the entire history of the system. To address the non-Markovian dynamics, an LSTM component is added to the state representation $\mathbf{s}_t$ to capture the temporal dependencies in the data. This approach is known as recurrent PPO (Ni et al., 2021). Let h_t be the hidden state of the LSTM at time t. The state representation $\mathbf{s}_t$ is fed into the LSTM at each time step to update the hidden state h_t.

$$\mathbf{s}_t = (S_t, F_t, G_t, \tau, \Delta_{t-1}), \quad h_t = \text{LSTM}(\mathbf{s}_t, h_{t-1}) \tag{6.17}$$

In a LSTM, the cell state c_t stores a representation of historical information up to time t. It is updated at each time step by a forget gate f_t, an input gate i_t, and an output gate o_t.

In a recurrent PPO with LSTM, the policy and reward function remain the same but with the new state representation in Equation (6.17). Instead of following the standard PPO algorithm, the agent uses the recurrent PPO algorithm to learn an optimal policy for hedging VAs. Figure 6.1 illustrates the network architecture of the standard PPO and the recurrent PPO with LSTM, and Algorithm 10 outlines the recurrent PPO algorithm and for training an agent to hedge VAs with non-Markovian dynamics.

Algorithm 10 Recurrent PPO with LSTM for Hedging Variable Annuities

1: Initialize policy parameters θ_0 and value function parameters ϕ_0.
2: Initialize the LSTM layers with random weights, and set the hidden state h_0 to zeros.
3: **For** iteration $k = 0, 1, 2, \ldots$ **do**
4: Collect a set of trajectories $\mathcal{D}_k$ by sampling from the environment with policy π_{θ_k}.
5: Reset the LSTM hidden state h_0 to zeros at the beginning of each trajectory.
6: Compute the rewards $\hat{R}_t$ for each trajectory in $\mathcal{D}_k$.
7: Estimates the advantage function with $\hat{A}_{\pi_{\theta_k}}(\mathbf{s}_t, a_t)$ using value function V_{ϕ_k}.
8: Update the policy parameters θ_{k+1} by

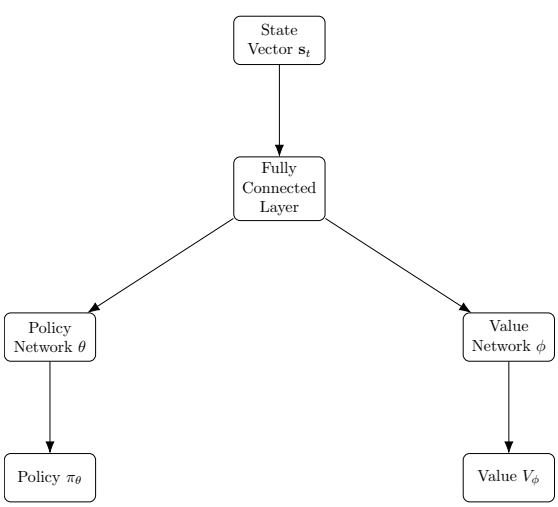
$$\max_{\theta} \mathbb{E}\left[\min\left\{\Lambda_t(\theta)\hat{A}^{\pi_\theta}(\mathbf{s}_t, a_t), \text{clip}(\Lambda_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_\theta}(\mathbf{s}_t, a_t)\right\}\right]$$

9: Update the value function parameters ϕ_{k+1} by minimizing the mean squared error between the predicted value and the actual collected reward.

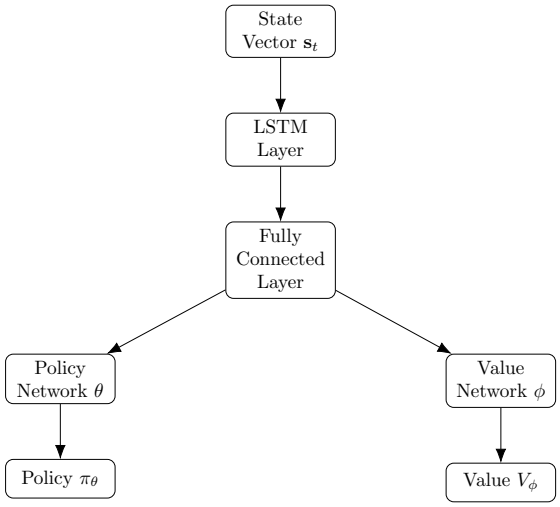
$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\kappa \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(\mathbf{s}_t) - \hat{R}_t\right)^2$$

10: **End For**

The recurrent PPO algorithm is similar to the standard PPO algorithm, but it includes an LSTM component to capture the temporal dependencies in the data. However, the sequantial nature of the LSTM is not suitable for a parallel computation, which slows down the training process.



(a) regular PPO



(b) Recurrent PPO with LSTM

Figure 6.1: PPO Network Architectures

6.4 Transfer Learning for Hedging VAs

In the context of deep hedging, various research has been conducted to improve the hedging strategies on real-world data by leveraging knowledge from simulated data. Most of the research uses the offline-online learning approach, where the hedging strategies are first trained with simulation data, but updated in real-time based on new market information and continuous learning to adapt to changing market dynamics. The most relevant paper to our study is Xiao et al. (2021), which use policy gradient algorithms to train hedging strategies for European options with simulation data from Heston model and then updated in real-time hedging S&P 500 index options. To the best of our knowledge, no research has been conducted to improve the hedging strategies with the state-of-the-art transfer learning techniques. This section introduces some transfer learning techniques that are relevant to our task of hedging VAs.

6.4.1 Reward Shaping

In the context of hedging VAs using deep reinforcement learning and transfer learning, reward shaping becomes a crucial technique. It involves modifying the reward function in the target domain to incorporate additional guidance, which often leads to more efficient

learning and a better performance. This technique can be particularly useful for managing the complex risks associated with VAs, as it can guide the learning process towards strategies that are more effective in hedging these products.

One way to implement reward shaping in this context is by introducing auxiliary rewards that reflect the performance of hedging strategies learned from the source domains. Reward shaping learns an auxiliary reward function $\mathcal{R}_s : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ from the source domains. The auxiliary reward function is then used to shape the reward function in the target domain, $\mathcal{R}'_t = \mathcal{R}_t + \mathcal{R}_s$, by incorporating the insights gained from the source domains.

Some popular reward shaping techniques include potential-based reward shaping (Ng et al., 1999), potential-based state-action advice (Wiewiora et al., 2003), dynamic value function advice (Harutyunyan et al., 2015).

6.4.2 Policy Transfer

The most straightforward approach to transfer learning is to directly transfer the policy learned from the source domains to the target domain via policy distillation. Policy distillation is a technique that transfers the knowledge from a teacher policy to a student policy by training the student policy to mimic the teacher policy up to a certain extent. The teacher policy is learned from the source domains, and the student policy learns the hedging strategies in the target domain while guided by the teacher policy. In the Distral algorithm (Teh et al., 2017), the student policy maximizes a multi-task objective: $\max_\phi \sum_{i=1}^{K} \mathcal{J}(\pi_\phi, \pi_{E_i})$, where

$$\mathcal{J}(\pi_\phi, \pi_{E_i}) = \sum_{t=0}^{\infty} \mathbb{E}_{s_t \sim \mu_0^t, a_t \sim \pi_\phi} \left[\gamma^t R_{t+1} + \frac{\alpha}{\beta} \log \pi_\phi(a_t|s_t) - \frac{1}{\beta} log \pi_{E_i}(a_t|s_t) \right], \tag{6.18}$$

where a set of K teacher policies π_{E_i} are learned from the source domains, and the student policy π_ϕ is learned on the target domain. α and β are hyperparameters that control the trade-off between mimicking the teacher policies and exploring the target domain.

6.4.3 Evaluation Metrics

The metrics discussed for evaluating transfer learning approaches focus on two key aspects: mastery and generalization. Mastery assesses the agent's final performance level

in the target domain, indicating how effectively it has learned a specific task from previous knowledge. Some common metrics for evaluating mastery include: Generalization, on the other hand, refers to the agent's ability to quickly adapt its learned knowledge to the target domain. In the context of risk management, generalization is more important than mastery, as the agent must be able to quickly adapt to changing market conditions and new financial products. Some common metrics for evaluating generalization include:

- **Jumpstart performance:** the initial reward of the agent in the target domain.
- **Accumulated reward:** the total area under the reward curve over time in the target domain.
- **Time to threshold:** the time it takes for the agent to reach a certain performance threshold in the target domain.
- **Performance with fixed epochs:** the agent's performance in the target domain after a fixed number of epochs.

6.5 Numerical Experiments

We present a numerical experiment to demonstrate the effectiveness of the model-free PPO algorithm for hedging VAs with transaction costs. We consider a VA contract with a maturity of $T = 240$ months and a GMMB rider with a simulation environment described in Section 6.1.1. The simulation enviornment is implemented in Python using the OpenAI Gym framework (Brockman et al., 2016), and the neural networks are programmed using the PyTorch library (Paszke et al., 2019). The environment is designed to simulate the dynamics of the VA contract and provide the agent with the current state of the VA with the contract specifications and the financial market information described in Table 6.1.

The underlying asset price S_t is modeled as a GBM with a drift rate μ and a volatility σ.

Figure 6.2 illustrates the hedging performance of the deep hedging algorithm with transfer learning for hedging VAs with a GMMB rider. The first deep hedging agent is directly trained on the Heston asset model, and the second deep hedging agent is trained with transfer learning from the GBM model. The figure plot the 95%-CVaR of hedging errors against training epochs for both agents. The deep hedging agent with transfer learning outperforms the deep hedging agent trained on the Heston model in both mastery

Hyperparameter	Value
$\lvert\mathcal{D}_k\rvert$	16
ϵ	0.2
Learning rate	0.0001
S_0	1000
μ	0.00375
σ	0.0457627
r	0.002
T	240

Table 6.1: PPO Hyperparameters and GMMB Contract Specifications

and generalization. The agent with transfer learning achieves a lower loss in the target domain and adapts more quickly to the new environment, indicating that the agent has learned a more robust hedging strategy that can be effectively applied to a more complex environment.

For the PPO agent, we use a regular PPO and a recurrent PPO with an LSTM component to capture the temporal dependencies in the data. The weights of the value network and the policy network are initialized with orthogonal initialization (Engstrom et al., 2020) with scaling $\sqrt{2}$ and the bias terms set to zero. The value network and the policy network are implemented as fully connected neural networks with two hidden layers of 16 units each and tanh activation functions. They are trained using the Adam optimizer with a learning rate of 0.0001 and a batch size automatically adjusted to the number of trajectories in the collected dataset. At each timestep, the observations are normalized and cliped between -10 and 10 before being fed into the neural networks to stabilize the training process. For PPO, we follow the suggestions in Schulman et al. (2017) and set the clipping hyperparameter ϵ to 0.2 to prevent large policy updates. More specifically, our implementation of the PPO uses a generalized advantage estimation (Schulman et al., 2015b) to estimate the advantage function. The agent interacts with the environment for 10^5 episodes, and the policy and value networks are evaluated every 1000 episodes. The performance of the agent is evaluated based on the average reward over an independent test set of 10^3 episodes, and the results are compared with the benchmark hedging strategy, delta hedging.

This experiment compares the hedging performance of the recurrent PPO agent with LSTM and the deep hedging algorithm proposed by Buehler et al. (2019) for hedging VAs with a GMMB rider. The experiment is conducted with the same amount ofsimulation budget.

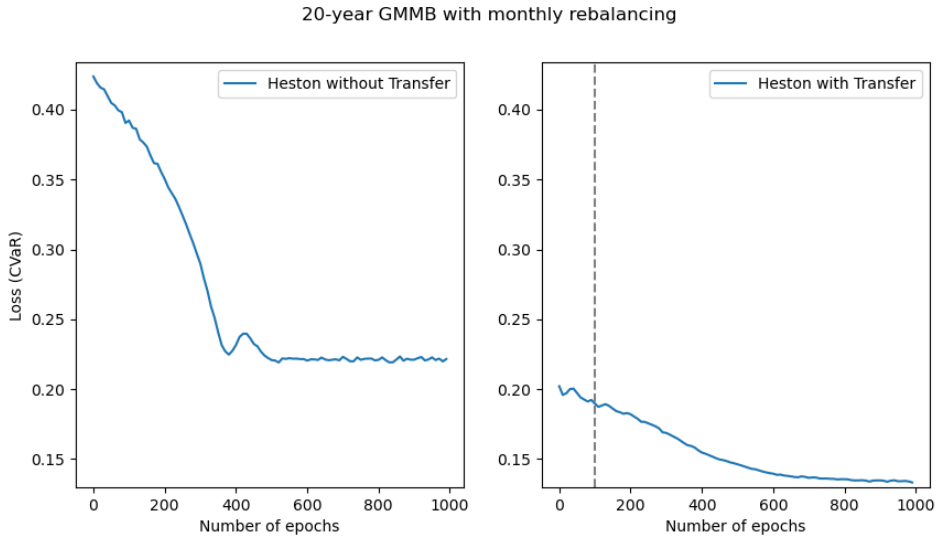


Figure 6.2: Hedging performance of deep hedging with transfer learning

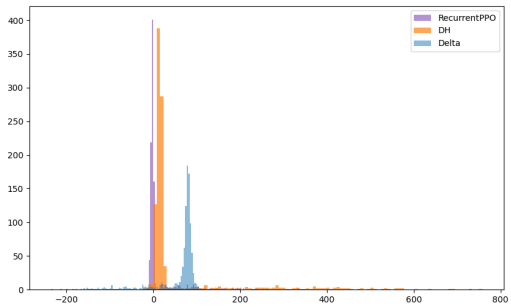
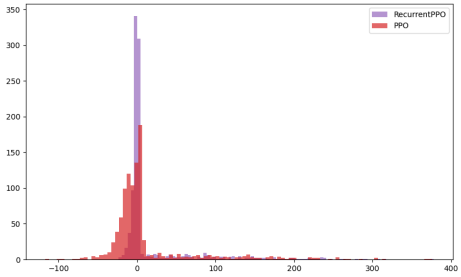
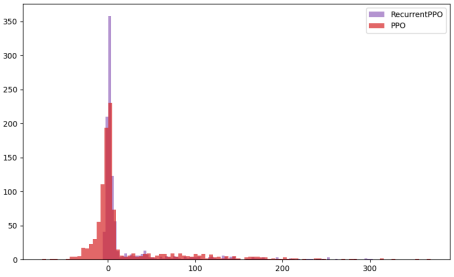


Figure 6.3: Hedging performance of recurrent PPO and deep hedging

Figure 6.4 illustrates the hedging performance of the standard PPO and the recurrent PPO with LSTM for hedging VAs with transaction costs $C = 0.005$. The figures show the distributions of the hedging errors incurred by the agents over the 10^3 test episodes for the GBM and the RS-GBM asset models. For both asset models, the recurrent PPO outperforms the standard PPO in terms of hedging performance. For the regime-switching GBM, the 95%-VaR of hedging errors of the recurrent PPO and the standard PPO are 136.24 and 147.14, respectively. The hedging errors of the recurrent PPO are more concentrated around zero, indicating that historical information captured by the LSTM component helps



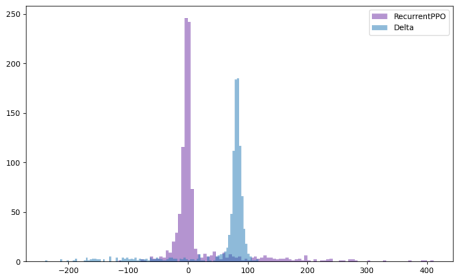
(a) GBM



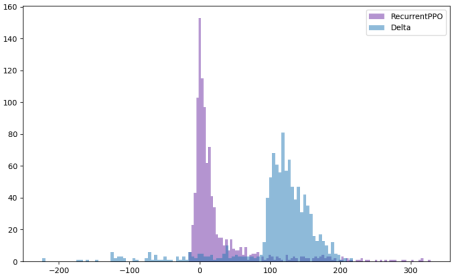
(b) Regime-switching GBM

Figure 6.4: Hedging performance of standard PPO and recurrent PPO

the agent make better hedging decisions even in the enviornment is Markovian.



(a) $C = 0.001$



(b) $C = 0.02$

Figure 6.5: Hedging performance of recurrent PPO and delta hedging with different transaction costs

Figure 6.5 shows the hedging performance of the recurrent PPO agent and the delta hedging strategy with different transaction costs when the asset model is a GBM. The experiment is conducted with transaction costs of $C = 0.001$ and $C = 0.02$. The delta hedging strategy is implemented with the Black-Scholes delta. When the transaction cost is low, the delta hedging strategy outperforms the recurrent PPO agent. However, as the transaction cost increases, delta hedging becomes less effective, and the recurrent PPO agent is able to learn from the environment and adapt its hedging strategy to minimize the hedging errors.

Figure 6.6 illustrates the hedging performance of the recurrent PPO agent with and

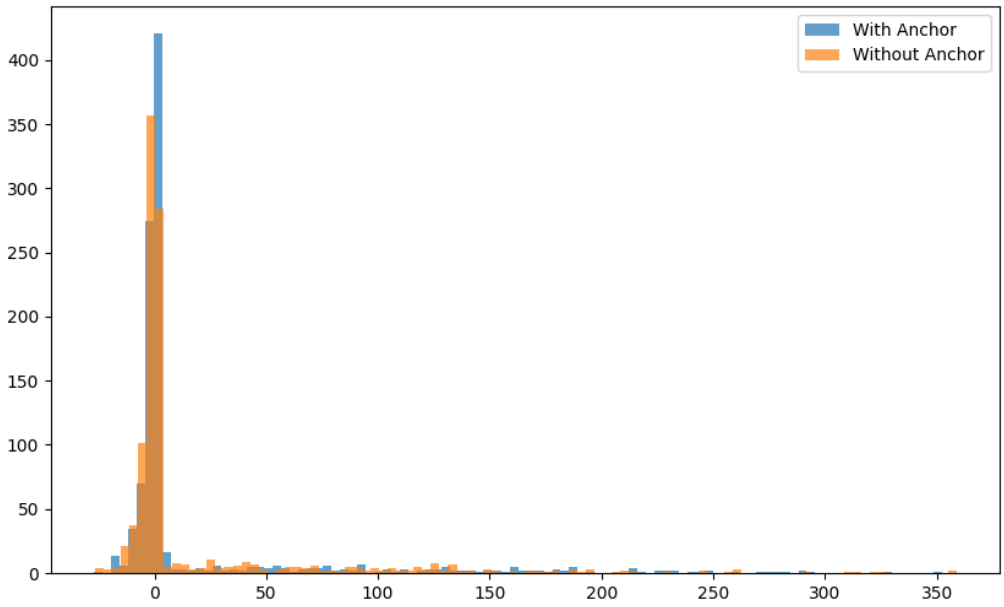


Figure 6.6: Hedging performance of recurrent PPO with or without anchor

without the model information of the underlying asset model and the current value of the liability. The experiment is conducted for the GBM asset model with transaction costs of $C = 0.005$. The results show that the agent with the model information does not outperform the agent without the model information.

Figure 6.7 illustrates the hedging performance of the recurrent PPO agent for hedging VAs with a GMWB rider. The experiment is conducted with the same hyperparameters as the experiment with the GMMB rider. Unlike the GMMB rider, the PPO agent struggles to learn an optimal hedging strategy for the VA with a GMWB rider. The hedging errors of the PPO agent are more dispersed, indicating that more training episodes are needed to achieve a satisfactory hedging performance. The results suggest that the GMWB rider introduces additional complexity to the hedging problem, and the model-free RL algorithm is not sample efficient.

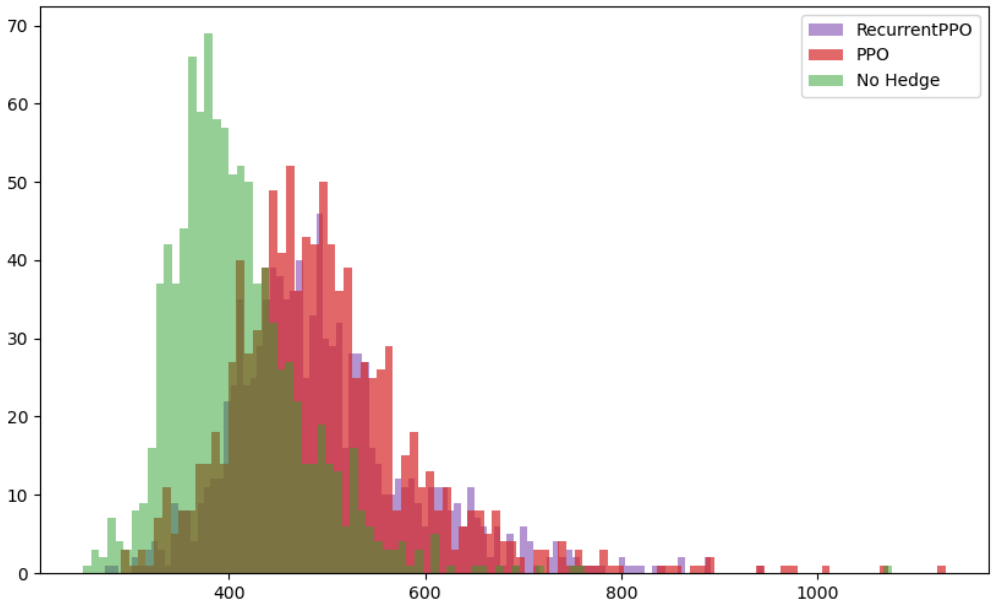


Figure 6.7: Hedging performance of recurrent PPO with GMWB

6.6 Future Directions

The proposed future work of this thesis aims at investigating how transfer learning can be effectively integrated into the training of PPO agents for dynamic hedging of VA contracts. The primary goal is to enhance the adaptability and efficiency of hedging strategies in response to new VA products and changing market conditions.

The first objective is to establish a systematic approach for transferring policies learned by PPO agents from source tasks to target tasks in the context of dynamic hedging. Source tasks may involve hedging a GMMB under a specific market model, while target tasks could entail hedging a GMWB under a different market model.

To achieve this, the research will explore methodologies such as policy distillation, where knowledge from a pre-trained "teacher" policy is transferred to a "student" policy (Rusu et al., 2015). Policy distillation has been effective in compressing and transferring knowledge between networks, improving learning efficiency in RL settings. Another avenue for future research is the use of pre-trained model initialization, wherein the policy network

trained on the source task initializes the policy for the target task. This method leverages the shared underlying structures between tasks, which can potentially accelerate convergence and improve performance. The framework will also consider feature representation transfer, which aims at reusing learned representations of market dynamics and policyholder behaviors between tasks. By capturing common patterns across different contracts and market models, the PPO agent can more effectively generalize to new scenarios (Bengio, 2012).

By addressing these objectives, the research aims at advancing the field of financial risk management through the development of adaptive and efficient hedging strategies for VA contracts. The integration of transfer learning into PPO agents has the potential to substantially reduce computational costs and improve the robustness of hedging policies in dynamic market environments.

References

Ahmed, M., Mahmood, A. N., and Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31.

Andreas, J., Klein, D., and Levine, S. (2017). Modular multitask reinforcement learning with policy sketches. In *International conference on machine learning*, pages 166–175. PMLR.

Artzner, P., Delbaen, F., Eber, J.-M., and Heath, D. (1999). Coherent measures of risk. *Mathematical Finance*, 9(3):203–228.

Barton, R. R. (1998). Simulation metamodels. In *1998 Winter Simulation Conference. Proceedings (Cat. No. 98CH36274)*, volume 1, pages 167–174. IEEE.

Bauer, D., Kling, A., and Russ, J. (2008). A universal pricing framework for guaranteed minimum benefits in variable annuities. *ASTIN Bulletin: The Journal of the IAA*, 38(2):621–651.

Bauer, D., Reuss, A., and Singer, D. (2012). On the calculation of the solvency capital requirement based on nested simulations. *ASTIN Bulletin: The Journal of the IAA*, 42(2):453–499.

Bellman, R. (1966). Dynamic programming. *science*, 153(3731):34–37.

Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36. JMLR Workshop and Conference Proceedings.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer.

Bollerslev, T. (1990). Modelling the coherence in short-run nominal exchange rates: a multivariate generalized arch model. *The review of economics and statistics*, pages 498–505.

Box, G. E. and Pierce, D. A. (1970). Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association*, 65(332):1509–1526.

Boyle, P. and Hardy, M. (2003). Guaranteed annuity options. *ASTIN Bulletin: The Journal of the IAA*, 33(2):125–152.

Boyle, P. P. and Hardy, M. R. (1997). Reserving for maturity guarantees: Two approaches. *Insurance: Mathematics and Economics*, 21(2):113–127.

Boyle, P. P. and Schwartz, E. S. (1977). Equilibrium prices of guarantees under equity-linked contracts. *Journal of Risk and Insurance*, 44(4):639–660.

Broadie, M., Du, Y., and Moallemi, C. C. (2015). Risk estimation via regression. *Operations Research*, 63(5):1077–1097.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Brown, T. B. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Buehler, H., Gonon, L., Teichmann, J., and Wood, B. (2019). Deep hedging. *Quantitative Finance*, 19(8):1271–1291.

Carbonneau, A. (2021). Deep hedging of long-term financial derivatives. *Insurance: Mathematics and Economics*, 99:327–340.

Carlini, N. and Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee.

Caruana, R. (1997). Multitask learning. *Machine learning*, 28:41–75.

Caruana, R., Lawrence, S., and Giles, C. (2000). Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, 13.

Cathcart, M. J., Lok, H. Y., McNeil, A. J., and Morrison, S. (2015). Calculating variable annuity liability "greeks" using monte carlo simulation. *ASTIN Bulletin: The Journal of the IAA*, 45(2):239–266.

Chen, P., Lezmi, E., Roncalli, T., and Xu, J. (2020). A note on portfolio optimization with quadratic transaction costs. *arXiv preprint arXiv:2001.01612*.

Cheng, X., Luo, W., Gan, G., and Li, G. (2019). Fast valuation of large portfolios of variable annuities via transfer learning. In *PRICAI 2019: Trends in Artificial Intelligence: 16th Pacific Rim International Conference on Artificial Intelligence, Cuvu, Yanuca Island, Fiji, August 26-30, 2019, Proceedings, Part III 16*, pages 716–728. Springer.

Chong, W. F., Cui, H., and Li, Y. (2023). Pseudo-model-free hedging for variable annuities via deep reinforcement learning. *Annals of Actuarial Science*, 17(3):503–546.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*. https://arxiv.org/abs/1412.3555. Accessed 7th Sep 2023.

Cont, R. (2001). Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative finance*, 1(2):223.

Coppersmith, D. and Winograd, S. (1987). Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6.

Dang, O. (2021). *Efficient Nested Simulation of Tail Risk Measures for Variable Annuities*. Ph.D. thesis, Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, ON, Canada. https://uwspace.uwaterloo.ca/handle/10012/17084.

Dang, O., Feng, M., and Hardy, M. R. (2020). Efficient nested simulation for conditional tail expectation of variable annuities. *North American Actuarial Journal*, 24(2):187–210.

Dang, O., Feng, M., and Hardy, M. R. (2022). Dynamic importance allocated nested simulation for variable annuity risk measurement. *Annals of Actuarial Science*, 16(2):319–348.

Dang, O., Feng, M., and Hardy, M. R. (2023). Two-stage nested simulation of tail risk measurement: A likelihood ratio approach. *Insurance: Mathematics and Economics*, 108:1–24.

Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Du, J., Jin, M., Kolm, P. N., Ritter, G., Wang, Y., and Zhang, B. (2020). Deep reinforcement learning for option replication and hedging. *The Journal of Financial Data Science*, 2(4):44–57.

EIOPA (2014). The underlying assumptions in the standard formula for the solvency capital requirement calculation. Technical report, The European Insurance and Occupational Pensions Authority.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Embrechts, P., Klüppelberg, C., and Mikosch, T. (2013). *Modelling extremal events: for insurance and finance*, volume 33. Springer Science & Business Media.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2020). Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*.

Feng, M. and Song, E. (2020). Optimal nested simulation experiment design via likelihood ratio method. *arXiv preprint arXiv:2008.13087*.

Feng, R., Cui, Z., and Li, P. (2016). Nested stochastic modeling for insurance companies. Technical report, Society of Actuaries.

Feng, R., Gan, G., and Zhang, N. (2022). Variable annuity pricing, valuation, and risk management: A survey. *Scandinavian Actuarial Journal*, 2022(10):867–900.

Feng, R. and Jing, X. (2017). Analytical valuation and hedging of variable annuity guaranteed lifetime withdrawal benefits. *Insurance: Mathematics and Economics*, 72:36–48.

Fonseca, D. J., Navaresse, D. O., and Moynihan, G. P. (2003). Simulation metamodeling through artificial neural networks. *Engineering applications of artificial intelligence*, 16(3):177–183.

Frazier, P. I. (2018). Bayesian optimization. In *Recent advances in optimization and modeling of contemporary problems*, pages 255–278. Informs.

Galton, F. (1886). Regression towards mediocrity in hereditary stature. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 15:246–263.

Gan, G. (2013). Application of data clustering and machine learning in variable annuity valuation. *Insurance: Mathematics and Economics*, 53(3):795–801.

Gan, G. and Lin, S. X. (2015). Valuation of large variable annuity portfolios under nested simulation: A functional data approach. *Insurance: Mathematics and Economics*, 62:138–150.

Gârleanu, N. and Pedersen, L. H. (2013). Dynamic trading with predictable returns and transaction costs. *The Journal of Finance*, 68(6):2309–2340.

Gatheral, J., Jaisson, T., and Rosenbaum, M. (2022). Volatility is rough. In *Commodities*, pages 659–690. Chapman and Hall/CRC.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

Giles, M. B. (2015). Multilevel monte carlo methods. *Acta numerica*, 24:259–328.

Giles, M. B. and Haji-Ali, A.-L. (2019). Multilevel nested simulation for efficient risk estimation. *SIAM/ASA Journal on Uncertainty Quantification*, 7(2):497–525.

Giurca, A. and Borovkova, S. (2021). Delta hedging of derivatives using deep reinforcement learning. *Available at SSRN 3847272*.

Glasserman, P. (2004). *Monte Carlo methods in financial engineering*, volume 53. Springer.

Golestaneh, P., Taheri, M., and Lederer, J. (2024). How many samples are needed to train a deep neural network? *arXiv preprint arXiv:2405.16696*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016a). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Goodfellow, I., Bengio, Y., and Courville, A. (2016b). *Deep Learning*. MIT Press.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*. https://arxiv.org/abs/1412.6572. Accessed 1st Jan 2024.

Gordy, M. B. and Juneja, S. (2010). Nested simulation in portfolio risk measurement. *Management Science*, 56(10):1833–1848.

Ha, H. and Bauer, D. (2015). A least-squares monte carlo approach to the calculation of capital requirements. In *World Risk and Insurance Economics Congress*, volume 6, pages 2–6. https://danielbaueracademic.files.wordpress.com/2013/11/bauerha_2015.pdf. Accessed 7th Sep 2023.

Hamilton, J. D. (1989). A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica: Journal of the econometric society*, pages 357–384.

Härdle, W. (1990). *Applied nonparametric regression*. Cambridge university press.

Hardy, M. R. (2001). A regime-switching model of long-term stock returns. *North American Actuarial Journal*, 5(2):41–53.

Hardy, M. R. (2003). *Investment Guarantees: Modeling and Risk Management for Equity-Linked Life Insurance*, volume 168. John Wiley & Sons.

Hardy, M. R. (2006). An introduction to risk measures for actuarial applications. *SOA Syllabus Study Note*, 19:9–14.

Hardy, M. R. and Saunders, D. (2022). *Quantitative enterprise risk management*. Cambridge University Press.

Harutyunyan, A., Devlin, S., Vrancx, P., and Nowé, A. (2015). Expressing arbitrary reward functions as potential-based advice. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29.

Hasbrouck, J. (1991). Measuring the information content of stock trades. *The Journal of Finance*, 46(1):179–207.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

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

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hong, J. L., Hu, Z., and Liu, G. (2014). Monte carlo methods for value-at-risk and conditional value-at-risk: A review. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 24(4):1–37.

Hong, J. L., Juneja, S., and Liu, G. (2017). Kernel smoothing for nested estimation with application to portfolio risk measurement. *Operations Research*, 65(3):657–673.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Horvath, B., Teichmann, J., and Žurič, Ž. (2021). Deep hedging under rough volatility. *Risks*, 9(7):138.

Hull, J. C. and Basu, S. (2016). *Options, futures, and other derivatives*. Pearson Education India.

Imaki, S., Imajo, K., Ito, K., Minami, K., and Nakagawa, K. (2021). No-transaction band network: A neural network architecture for efficient deep hedging. *arXiv preprint arXiv:2103.01775*.

Jabbar, H. and Khan, R. Z. (2015). Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices*, 70(10.3850):978–981.

Jennen-Steinmetz, C. and Gasser, T. (1988). A unifying approach to nonparametric regression estimation. *Journal of the American Statistical Association*, 83(404):1084–1089.

Jeong, G. and Kim, H. Y. (2019). Improving financial trading decisions using deep q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications*, 117:125–138.

Jiang, L., Huang, D., Liu, M., and Yang, W. (2020). Beyond synthetic noise: Deep learning on controlled noisy labels. In *International conference on machine learning*, pages 4804–4815. PMLR.

Jin, Z. L., Liu, Y., and Durlofsky, L. J. (2020). Deep-learning-based surrogate model for reservoir simulation with time-varying well controls. *Journal of Petroleum Science and Engineering*, 192:107273.

Kim, P.-S. and Kutzner, A. (2008). Ratio based stable in-place merging. In *International Conference on Theory and Applications of Models of Computation*, pages 246–257. Springer.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. https://arxiv.org/pdf/1412.6980.pdf. Accessed 7th Sep 2023.

Kleijnen, J. P. (2018). *Design and analysis of simulation experiments*. Springer.

Kolm, P. N. and Ritter, G. (2019). Dynamic replication and hedging: A reinforcement learning approach. *The Journal of Financial Data Science*, 1(1):159–171.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Lebichot, B., Verhelst, T., Le Borgne, Y.-A., He-Guelton, L., Oble, F., and Bontempi, G. (2021). Transfer learning strategies for credit card fraud detection. *IEEE access*, 9:114754–114766.

LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lieu, Q. X., Nguyen, K. T., Dang, K. D., Lee, S., Kang, J., and Lee, J. (2022). An adaptive surrogate model to structural reliability analysis using deep neural network. *Expert Systems with Applications*, 189:116104.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321.

Lin, X. S. and Yang, S. (2020a). Efficient dynamic hedging for large variable annuity portfolios with multiple underlying assets. *ASTIN Bulletin: The Journal of the IAA*, 50(3):913–957.

Lin, X. S. and Yang, S. (2020b). Fast and efficient nested simulation for large variable annuity portfolios: A surrogate modeling approach. *Insurance: Mathematics and Economics*, 91:85–103.

Liu, M. and Staum, J. (2010). Stochastic kriging for efficient nested simulation of expected shortfall. *Journal of Risk*, 12(3):3.

Longstaff, F. A. and Schwartz, E. S. (2001). Valuing american options by simulation: A simple least-squares approach. *The Review of Financial Studies*, 14(1):113–147.

Luo, W., Li, Y., Urtasun, R., and Zemel, R. (2016). Understanding the effective receptive field in deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 29.

Mack, Y.-P. (1981). Local properties of k-nn regression estimates. *SIAM Journal on Algebraic Discrete Methods*, 2(3):311–323.

Malmsten, H. and Teräsvirta, T. (2010). Stylized facts of financial time series and three popular models of volatility. *European Journal of pure and applied mathematics*, 3(3):443–477.

Marshall, C., Hardy, M., and Saunders, D. (2010). Valuation of a guaranteed minimum income benefit. *North American Actuarial Journal*, 14(1):38–58.

Marukame, T., Ueyoshi, K., Asai, T., Motomura, M., Schmid, A., Suzuki, M., Higashi, Y., and Mitani, Y. (2016). Error tolerance analysis of deep learning hardware using a restricted boltzmann machine toward low-power memory implementation. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 64(4):462–466.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Moerland, T. M., Broekens, J., Plaat, A., Jonker, C. M., et al. (2023). Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118.

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142.

NAIC (2021). *Valuation Manual*. National Association of Insurance Commissioners, Washington, DC, jan. 1 edition. https://content.naic.org.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., and Martens, J. (2015). Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*. https://arxiv.org/abs/1511.06807. Accessed 1st Jan 2024.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer.

Ni, T., Eysenbach, B., and Salakhutdinov, R. (2021). Recurrent model-free rl can be a strong baseline for many pomdps. *arXiv preprint arXiv:2110.05038*.

Nyström, E. J. (1930). Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Mathematica*.

OpenAI (2023). Chatgpt. https://chat.openai.com/chat. Accessed 7th Sep 2023.

OSFI (2017). Life insurance capital adequacy test. Technical report, Office of the Superintendent of Financial Institutes Canada.

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Peng, Y. and Nagata, M. H. (2020). An empirical overview of nonlinearity and overfitting in machine learning using covid-19 data. *Chaos, Solitons & Fractals*, 139:110055.

Piscopo, G. and Haberman, S. (2011). The valuation of guaranteed lifelong withdrawal benefit options in variable annuity contracts and the impact of mortality risk. *North American Actuarial Journal*, 15(1):59–76.

Poole, B., Sohl-Dickstein, J., and Ganguli, S. (2014). Analyzing noise in autoencoders and deep networks. *arXiv preprint arXiv:1406.1831*. https://arxiv.org/abs/1406.1831. Accessed 1st Jan 2024.

Prechelt, L. (2002). Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Rockafellar, R. T. and Uryasev, S. (2002). Conditional value-at-risk for general loss distributions. *Journal of Banking & Finance*, 26(7):1443–1471.

Rosen, S. L., Saunders, C. P., and Guharay, S. K. (2012). Metamodeling of simulations consisting of time series inputs and outputs. In *Proceedings of the 2012 Winter Simulation Conference (WSC)*, pages 1–12. IEEE.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386.

Ruf, J. and Wang, W. (2022). Hedging with linear regressions and neural networks. *Journal of Business & Economic Statistics*, 40(4):1442–1454.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California University San Diego La Jolla Institute for Cognitive Science.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*.

Salle, I. and Yıldızoğlu, M. (2014). Efficient eampling and meta-modeling for computational economic models. *Computational Economics*, 44:507–536.

Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Seber, G. A. and Lee, A. J. (2012). *Linear regression analysis*. John Wiley & Sons.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. (2015). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Stothers, A. J. (2010). On the complexity of matrix multiplication. *Edinburgh Research Archive*.

Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Szeg, G. (1939). *Orthogonal polynomials*, volume 23. American Mathematical Soc.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*. https://arxiv.org/abs/1312.6199. Accessed 1st Jan 2024.

Tang, M., Liu, Y., and Durlofsky, L. J. (2020). A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems. *Journal of Computational Physics*, 413:109456.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30.

The Geneva Association (2013). Variable annuities—an analysis of financial stability. Technical report, The Geneva Association.

Torres-Huitzil, C. and Girau, B. (2017). Fault and error tolerance in neural networks: A review. *IEEE Access*, 5:17322–17341.

Wang, W., Wang, Y., and Zhang, X. (2022). Smooth nested simulation: bridging cubic and square root convergence rates in high dimensions. *arXiv preprint arXiv:2201.02958*.

Watson, G. S. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372.

Wiewiora, E., Cottrell, G. W., and Elkan, C. (2003). Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 792–799.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Wirch, J. L. and Hardy, M. R. (1999). A synthesis of risk measures for capital adequacy. *Insurance: Mathematics and Economics*, 25(3):337–347.

Xiao, B., Yao, W., and Zhou, X. (2021). Optimal option hedging with policy gradient. In *2021 International Conference on Data Mining Workshops (ICDMW)*, pages 1112–1119. IEEE.

Xu, W. and Dai, B. (2022). Delta-gamma-like hedging with transaction cost under reinforcement learning technique. *The Journal of Derivatives*, 29(5):60–82.

Xu, X. (2020). *Variable annuity guaranteed benefits: An integrated study of financial modelling, actuarial valuation and deep learning*. PhD thesis, UNSW Sydney.

Yan, P., Abdulkadir, A., Luley, P.-P., Rosenthal, M., Schatte, G. A., Grewe, B. F., and Stadelmann, T. (2024). A comprehensive survey of deep transfer learning for anomaly detection in industrial time series: Methods, applications, and directions. *IEEE Access*.

Yang, L., Bankman, D., Moons, B., Verhelst, M., and Murmann, B. (2018). Bit error tolerance of a cifar-10 binarized convolutional neural network processor. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. Institute of Electrical and Electronics Engineers.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27.

Zhang, A., Satija, H., and Pineau, J. (2018). Decoupling dynamics and reward for transfer learning. *arXiv preprint arXiv:1804.10689*.

Zhang, K., Feng, M., Liu, G., and Wang, S. (2022). Sample recycling for nested simulation with application in portfolio risk measurement. *arXiv preprint arXiv:2203.15929*.

Zhang, K., Liu, G., and Wang, S. (2021). Bootstrap-based budget allocation for nested simulation. *Operations Research*.