Efficient Nested Simulation of Tail Risk Measures with Machine Learning Proxies

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, 2022

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
Assistant 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

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.

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

Abstract

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 proxy model. In this study, we review the literature on nested simulation procedures in financial engineering and establish fair comparisons of different proxy models using the same set of numerical examples. Our study shows interesting findings on the performance of different proxy models and provides useful insights for practitioners to choose proxy models for nested simulation procedures.

Acknowledgements

I would like to thank Professor Ben Feng and Professor Tony Wirjanto for they valuable support. They have patiently helped me through my journey as a Ph.D. student.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Quantitative risk management is a key component of modern financial systems, ensuring the stability and resilience of markets, institutions, and portfolios against an array of risks. In the era of complex financial products, such as option portfolios and variable annuity contracts, traditional risk assessment methods often fall short in accurately capturing the multifaceted nature of market, credit, and operational risks. This is where advanced simulation techniques, particularly nested simulation procedures, become indispensable.

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* is referred 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 is referred to as the *outer scenarios*. For each outer scenario, an inner level simulation model generates scenario-wise samples of the portfolio losses, which is 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 has to make a trade-off between the number of outer scenarios and the number of inner replications. **?** are the first to analyze and propose the optimal budget allocation of a standard nested simulation procedure. The term *standard* refers to using a standard Monte Carlo estimator, the sample mean of the inner replication to estimate a scenario-wise portfolio loss for an outer scenario. **?** investigate the optimal budget allocation for a standard nested simulation procedure with respect to the mean squared error (MSE) of the estimated risk measure.

The standard nested simulation procedure is computationally expensive with a somewhat 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 the inner replications from other outer scenarios. This is referred to as pooling. Different methods pool in different ways, either by a trained proxy model or by pre-defined likelihood ratio weights. **?** propose a regression-based nested simulation procedure, which uses a trained regression proxy model to estimate the scenario-wise portfolio loss for an outer scenario by pooling the inner replications from all outer scenarios. For risk measures in certain forms, **?** show that it is optimal to allocate all simulation budget to the outer level simulation, and the inner replication should be kept to a minimum of 1. Similarly, **?**, **?**, and **?** use a kernel smoothing model, a likelihood ratio model, and a kernel ridge regression model as proxies to pool the inner replications from all outer scenarios. In simulation studies, this approach of using a model of a simulation model is known as metamodeling, and the models of a simulation model are also referred to as metamodels (**?**). Another line of research is the multi-level Monte Carlo (MLMC) method analyzed in **?**, 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. Many procedures are proposed in the literature, but they are not directly comparable due to different error metrics, different assumptions, and different numerical examples. Within a common analytical framework, we first summarize and compare their asymptotic rate of convergence. Their asymptotic convergence results are closely examined for their assumptions that guarantee the convergence. Furthermore, our study finds that 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 is the first of its kind to subject back all the aforementioned simulation procedures to a complete and unbiased comparison. Extensive numerical experiments are conducted to show, in practical examples, how well the finite-sample performance of a method matches its theoretical convergence behavior. With our numerical examples, we compare the nested simulation procedures for different payoff complexity, problem dimensions, and risk measures.

The rest of the paper is organized as follows. Section 2.2 introduces the nested simulation procedure and the standard Monte Carlo 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 the paper.

2.2 Problem Formulation

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

$$\rho(g(X)),$$

where $X \in \Omega$. $g(X)$ can't be directly evaluated, but it is the output of

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

Some common risk measures are in the nested expectation form, in which

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

where $h(\cdot)$ is a known function. Forms of h include the following:

- Smooth functions, e.g., a quadratic tracking error with benchmark b: $h(t) = (t-b)^2$
- Lipschitz continuous functions, e.g., a mean excess loss over threshold u: $h(t) = \max\{t-u, 0\}$. Here, h is a hockey-stick function.
- Indicator functions, e.g., probability of a large loss over a threshold u: $h(t) = \mathbb{I}_{\{t \geqslant u\}}$

Other risk measures of interest that are not in the nested expectation form are the value at risk (VaR) and the conditional value at risk (CVaR). The α-VaR of $g(X)$ is defined as

$$\text{VaR}_\alpha(g(X)) = q_\alpha = \inf\left\{q : \Pr(g(X) \leq q) \geq \alpha\right\}.$$

The α-CVaR of $g(X)$ is defined as

$$\text{VaR}_\alpha(g(X)) = \frac{1}{1-\alpha}\int_\alpha^1 q_v dv.$$

2.2.1 The Standard Nested Simulation Procedure

The standard nested simulation procedure first simulates M independent and identically distributed (iid) 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 iid. 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 $g(X_i)$ with a standard Monte Carlo estimator

$$\hat{g}_N(X_i) = \frac{1}{N} \sum_{j=1}^{N} Y_{ij}; \quad Y_{ij} \sim F_{Y|X_i}$$

Let $(\hat{g}_N(\mathbf{X}))_{[1]}, \ldots, (\hat{g}_N(\mathbf{X}))_{[M]}$ be the order statistics of $\hat{g}_N(X_1), \ldots \hat{g}_N(X_M)$. The standard nested simulation estimators for different forms of ρ are as follows:

1. Nested expectation form:

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

2. Value at risk (VaR):

$$\hat{\rho}_{M,N} = (\hat{g}_N(\mathbf{X}))_{\lceil \alpha M \rceil}$$

3. Conditional value at risk (CVaR):

$$\hat{\rho}_{M,N} = (\hat{g}_N(\mathbf{X}))_{\lceil \alpha M \rceil} + \frac{1}{(1-\alpha)M} \sum_{i=1}^{M} \max\{\hat{g}_N(X_i) - (\hat{g}_N(\mathbf{X}))_{\lceil \alpha M \rceil}, 0\}$$

? analyze the optimal budget allocation of the standard nested simulation procedure with respect to the MSE of the estimator $\hat{\rho}_{M,N}$.

2.2.2 Multi-level Monte Carlo

The multi-level Monte Carlo (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 Monte Carlo estimator has a high variance. The MLMC method estimates ρ with a multi-level Monte Carlo estimator:

$$\hat{\rho}_{\Gamma}^{\mathrm{MLMC}} = \sum_{\ell=0}^{L} \left(\frac{1}{M_\ell} \sum_{i=1}^{M_\ell} h(\hat{g}_{N_\ell}(X_{i,\ell})) - \frac{1}{M_{\ell-1}} \sum_{i=1}^{M_{\ell-1}} h(\hat{g}_{N_{\ell-1}}(X_{i,\ell-1})) \right), \quad X_{i,\ell} \sim F_X,$$

where $\hat{g}_N(\cdot) = 0$, L is the number of levels, M_ℓ is the number of outer scenarios at level ℓ, and N_ℓ is the number of inner replications at level ℓ. Applying the analysis of **?** in a nested simulation context, **?** show that the MLMC method can achieve a similar level of accuracy as the standard nested simulation procedure with a lower total computational budget. The simulation budget Γ is the sum of the computational budget at each level, that is, $\Gamma = \sum_{\ell=0}^{L} M_\ell \cdot N_\ell$.

2.2.3 Supervised Learning Models

In supervised learning, $g(\cdot)$ can be approximated by $\hat{g}_{M,N}^{\mathrm{SL}}(\cdot)$, which is based on a chosen function family $\mathcal{G}$ and observations from the standard nested simulation procedure. Consider the observation pairs $(X_i, \hat{g}_N(X_i))$ for $i \in \{1, \ldots, M\}$ as training data, we can use supervised learning to approximate $g(\cdot)$ by $\hat{g}_{M,N}^{\mathrm{SL}}(\cdot)$ and to pool the inner replications from all outer scenarios. Using the M *training* samples, a nested Monte Carlo estimator of ρ is given by

1. Nested expectation form:

$$\hat{\rho}_{M,N}^{\mathrm{SL,Train}} = \frac{1}{M} \sum_{i=1}^{M} h(\hat{g}_{M,N}^{\mathrm{SL}}(X_i)); \quad X_i \sim F_X$$

2. VaR:

$$\hat{\rho}_{M,N}^{\mathrm{SL,Train}} = (\hat{g}_{M,N}^{\mathrm{SL}}(\mathbf{X}))_{\lceil \alpha M \rceil}$$

3. CVaR:

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

where $(\hat{g}_{M,N}(\mathbf{X}))_{\lceil \alpha M \rceil}$ is the $\lceil \alpha M \rceil$-th order statistic of $\hat{g}^{\mathrm{SL}}_{M,N}(X_1), \ldots, \hat{g}_{M,N}(X_M)$. Similarly, with M' *test* samples of X, namely $\tilde{\mathbf{X}} = \tilde{X}_1, \ldots, \tilde{X}_{M'}$, an estimator is given by

1. Nested expectation form:

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

2. VaR:

$$\hat{\rho}^{\mathrm{SL,Test}}_{M,N,M'} = (\hat{g}^{\mathrm{SL}}_{M,N}(\tilde{\mathbf{X}}))_{\lceil \alpha M' \rceil}.$$

3. CVaR:

$$\begin{aligned}\hat{\rho}^{\mathrm{SL,Test}}_{M,N,M'} &= (\hat{g}^{\mathrm{SL}}_{M,N}(\tilde{\mathbf{X}}))_{\lceil \alpha M' \rceil} \\ &\quad + \frac{1}{(1-\alpha)M'}\sum_{i=1}^{M'} \max\{\hat{g}^{\mathrm{SL}}_{M,N}(\tilde{X}_i) - (\hat{g}^{\mathrm{SL}}_{M,N}(\tilde{\mathbf{X}}))_{\lceil \alpha M' \rceil}, 0\},\end{aligned}$$

where $(\hat{g}_{M,N}(\tilde{\mathbf{X}}))_{\lceil \alpha M' \rceil}$ is the $\lceil \alpha M' \rceil$-th order statistic of $\hat{g}^{\mathrm{SL}}_{M,N}(\tilde{X}_1), \ldots, \hat{g}_{M,N}(\tilde{X}_{M'})$. Note that $\hat{g}^{\mathrm{SL}}_{M,N}(\cdot)$ is derived from the training samples $(X_1, \hat{g}_N(X_1)), \ldots, (X_M, \hat{g}_N(X_M))$.

We are interested in minimizing the MSE of the supervised learning-based nested simulation estimator with supervised learning $\hat{\rho}^{\mathrm{SL,Train}}_{M,N}$ and $\hat{\rho}^{\mathrm{SL,Test}}_{M,N,M'}$ subject to the total simulation budget Γ.

$$\begin{aligned} \min_{M,N} \quad & \mathrm{MSE}(\hat{\rho}^{\mathrm{SL}}_{M,N}) = \mathbb{E}\left[\left(\hat{\rho}^{\mathrm{SL}}_{M,N} - \rho\right)^2\right] \\ \text{subject to} \quad & M \cdot N = \Gamma \end{aligned} \tag{2.1}$$

Existing literature on nested simulation procedures has proposed different methods to approximate the true function $g(\cdot)$ with supervised learning algorithms. Methods that include theoretical convergence results are regression (**?**), kernel smoothing (**?**), and kernel ridge regression (**?**). Their estimators of $g(\cdot)$ are given by $\hat{g}^{\mathrm{REG}}_{M,N}(\cdot)$, $\hat{g}^{\mathrm{KS}}_{M,N}(\cdot)$, and $\hat{g}^{\mathrm{KRR}}_{M,N}(\cdot)$, respectively.

- Regression:
$$\hat{g}^{\mathrm{REG}}_{M,N}(X) = \Phi(X)\hat{\beta},$$
where Φ is a chosen basis, and $\hat{\beta}$ is estimated from the training samples.

- Kernel smoothing:
$$\hat{g}^{\mathrm{KS}}_{M,N}(X) = \frac{\sum_{i=1}^{M} \bar{Y}_{N,i} K_w(X - X_i)}{\sum_{i=1}^{M} K_w(X - X_i)},$$
where K_w is the kernel function with bandwidth w.

- Kernel ridge regression:
$$\hat{g}^{\mathrm{KRR}}_{M,N}(X) = \underset{g \in \mathcal{N}_\Psi(\Omega)}{\arg\min} \left(\frac{1}{M} \sum_{i=1}^{M} (\hat{g}_N(X_i) - g(X_i))^2 + \lambda \|g\|^2_{\mathcal{N}_\Psi(\Omega)} \right),$$
where $\mathcal{N}_\Psi(\Omega)$ is the reproducing kernel Hilbert space (RKHS) with kernel Ψ defined domain Ω, and λ is the regularization parameter as in ridge regression. More specifically, Φ is a Matérn kernel with smoothness parameter ν and length scale parameter ℓ.

2.2.4 Likelihood Ratio Method

Instead of using a supervised learning model as proxy, **?** 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 whose 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

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

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

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

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

$$\begin{aligned} \min \quad & \mathbb{E}\left[(\hat{\rho}_{M,N} - \rho)^2\right] \\ \text{subject to} \quad & M \cdot N = \Gamma, \end{aligned} \tag{2.2}$$

where Γ is the total simulation budget for all nested simulation procedures considered in this study. A special case is the MLMC method, where the total simulation budget Γ is the sum of the computational budget at all levels.

2.3 Asymptotic Analysis

In 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.

Estimator for $g(X)$	**Smooth** h	**Lipschitz(hockey-stick** h**)**	**Indicator** h	**VaR**	**CVaR**
Standard Monte Carlo	$\star$	$\star(\checkmark)$	$\checkmark$	$\checkmark$	$\times$
Multi-level Monte Carlo	$\times$	$\times(\times)$	$\checkmark$	$\times$	$\times$
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 2.1: Existing asymptotic convergence results of nested simulation procedures for MSE

Table 2.1 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 × 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 $\Diamond$ indicates the asymptotic convergence result exists in the literature but in a weaker form. **?** provide the asymptotic convergence results for the standard nested simulation procedure with hockey-stick h, 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. **?** provide the asymptotic convergence results for the MLMC method with an indicator h. **?** provide the asymptotic convergence results for the regression-based nested simulation procedure with smooth h and 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. **?** and **?** provide the 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, **?** analyze the asymptotic convergence of the estimator of ρ in terms of the absolute error. Let $\hat{\rho}$ be the estimator of ρ. Its absolute error about ρ is defined as

$$\text{Absolute Error}\left(\hat{\rho}\right) = \left|\hat{\rho} - \rho\right|.$$

In **?**, the authors of the KRR-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 the absolute error. Instead of showing the convergence of the KRR-based estimator in terms of MSE as in **?**, we show the connections between the convergence in MSE and the convergence in probabilistic order for absolute error. Our findings in Section 2.3.1 show that the analysis of **?** indeed bridges the gap, but only in terms of the absolute error.

2.3.1 Connections between Convergence in MSE and Absolute Error

This section establishes the connections between the convergence in MSE and the convergence in probabilistic order for absolute error 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 absolute error, we first need to state the definition for a sequence of random variables to converge in those two forms.

Definition 1 *Let $\hat{\rho}_\Gamma$ be an estimator of ρ with a simulation budget of Γ. We write $\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right] = \mathcal{O}\left(\Gamma^{-k}\right)$, that is, $\hat{\rho}_\Gamma$ converges in MSE to ρ in order k if*

$$\exists C \limsup_M \frac{\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right]}{\Gamma^{-k}} \leq C$$

Definition 2 *Let $\hat{\rho}_\Gamma$ be an estimator of ρ with a simulation budget of Γ. We write $|\hat{\rho}_\Gamma - \rho| = \mathcal{O}_{\mathbb{P}}(\Gamma^{-k})$, that is $\hat{\rho}_\Gamma$ converges in probabilistic order k to ρ if for large enough Γ,*

$$\forall \epsilon > 0 \exists C \mathbb{P}\left(|\hat{\rho}_\Gamma - \rho| \geq C\Gamma^{-k}\right) \leq \epsilon$$

We start 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 $\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right] = \mathcal{O}\left(\Gamma^{-k}\right)$. Then, from the definition of convergence in MSE, there exists a constant C such that

$$\limsup_M \frac{\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right]}{\Gamma^{-k}} \leq C.$$

Hence, there exists some Γ such that for all $\gamma \geq \Gamma$,

$$\mathbb{E}\left[(\hat{\rho}_\gamma - \rho)^2\right] \leq C\gamma^{-k}.$$

The convergence in probabilistic order can be shown by separating the expectation into two parts: tail and non-tail.

$$\mathbb{E}\left[(\hat{\rho}_\gamma - \rho)^2 \cdot \mathbb{I}_{\{|\hat{\rho}_\gamma - \rho| \leq d\gamma^s\}}\right] + \mathbb{E}\left[(\hat{\rho}_\gamma - \rho)^2 \cdot \mathbb{I}_{\{|\hat{\rho}_\gamma - \rho| > d\gamma^s\}}\right] \leq C\gamma^{-k}.$$

The first term is always positive, and the second term can be bounded from below by the indicator function.

$$\begin{aligned}
\mathbb{E}\left[(\hat{\rho}_\gamma - \rho)^2 \cdot \mathbb{I}_{\{|\hat{\rho}_\gamma - \rho| > d\gamma^s\}}\right] &\geq \mathbb{E}\left[d^2\gamma^{2s} \cdot \mathbb{I}\{|\hat{\rho}_\gamma - \rho| > d\gamma^s\}\right] \\
&= d^2\gamma^{2s} \cdot \mathbb{E}\left[\mathbb{I}\{|\hat{\rho}_\gamma - \rho| > d\gamma^s\}\right] \\
&= d^2\gamma^{2s} \cdot \mathbb{P}\left(|\hat{\rho}_\gamma - \rho| > d\gamma^s\right)
\end{aligned}$$

Combining bounds on the two terms, we have

$$d^2\gamma^{2s}\mathbb{P}\left(|\hat{\rho}_\gamma - \rho| > d\gamma^s\right) \leq C\gamma^{-k}.$$

Let $s = -\frac{k}{2}$. Arranging the terms, we have

$$\mathbb{P}\left(|\hat{\rho}_\gamma - \rho| > d\gamma^{-\frac{k}{2}}\right) \leq \frac{C}{d^2}$$

Hence, for all $\epsilon > 0$, there exist $C^* = \sqrt{\frac{C}{\epsilon}}$ such that for all $\gamma \geq \Gamma$,

$$\mathbb{P}\left(|\hat{\rho}_\gamma - \rho| > C^*\gamma^{-\frac{k}{2}}\right) \leq \epsilon$$

In essence, the above is the definition of convergence in probabilistic order, that is,

$$|\hat{\rho}_\gamma - \rho| = \mathcal{O}_\mathbb{P}\left(\Gamma^{-\frac{k}{2}}\right)$$

Theorem 1 *Let $\hat{\rho}_\Gamma$ be an estimator of ρ with a simulation budget of Γ. If $\hat{\rho}_\Gamma$ converges in MSE to ρ in order k, then $\hat{\rho}_\Gamma$ converges in probabilistic order to ρ in order $\frac{k}{2}$.*

To the best of our knowledge, Theorem 1 has not been explicitly stated in the literature. It is the first result that shows the connections between the convergence in MSE and the convergence in probabilistic order for absolute error in the context of nested simulation. Theorem 1 is a general result that can be applied to any nested simulation procedure that converges in MSE to ρ in order k. If the estimator converges in MSE to ρ in order k, then it converges in probabilistic order to ρ in order $\frac{k}{2}$.

While the convergence in MSE implies the convergence in probabilistic order, the converse is not necessarily true. Similarly, the above argument is applied in reverse. Let $\hat{\rho}_\Gamma$ be an estimator of ρ with a simulation budget of Γ, and assume that $|\hat{\rho}_\Gamma - \rho| = \mathcal{O}_\mathbb{P}(\Gamma^{-k})$. The MSE of $\hat{\rho}_\Gamma$ can be separated into the same two parts.

$$\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right] = \mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2 \cdot \mathbb{I}_{\{|\hat{\rho}_\Gamma - \rho| \leq d\Gamma^{-2k}\}}\right] + \mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2 \cdot \mathbb{I}_{\{|\hat{\rho}_\Gamma - \rho| > d\Gamma^{-2k}\}}\right],$$

where the first term can be bounded from above.

$$\begin{aligned}\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2 \cdot \mathbb{I}_{\{|\hat{\rho}_\Gamma - \rho| > d\Gamma^s\}}\right] &\leq d^2\Gamma^{-2k} \cdot \mathbb{E}\left[\mathbb{I}\{|\hat{\rho}_\Gamma - \rho| > d\Gamma^s\}\right] \\ &= d^2\Gamma^{-2k} \cdot \mathbb{P}\left(|\hat{\rho}_\Gamma - \rho| > d\Gamma^s\right) \leq d^2\Gamma^{-2k}\end{aligned}$$

However, the second term is not always bounded. If the random variable $\hat{\rho}_\Gamma$ admits a density function f, then the second term can be further decomposed.

$$\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2 \cdot \mathbb{I}_{\{|\hat{\rho}_\Gamma - \rho| > d\Gamma^{-2k}\}}\right] = \int_{-\infty}^{-d\Gamma^{-2k}} (x-\rho)^2 f(x)dx + \int_{d\Gamma^{-2k}}^{\infty} (x-\rho)^2 f(x)dx$$

Hence, $\hat{\rho}_\Gamma$ converges in MSE to ρ in order $2k$ if and only if both integrals converge in order higher than $2k$. The above argument shows that the convergence in probabilistic order does not necessarily imply the convergence in MSE. The convergence in probabilistic order is a weaker form of convergence than the convergence in MSE.

2.3.2 Asymptotic Analysis for the Standard Nested Simulation Procedure

In **?**, 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 in the analysis of **?**.

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

Assumption 2 *$\hat{g}_N(X) = g(X) + \bar{Z}_N(X)$, where the simulation noise $\bar{Z}_N(X)$ has zero mean and variance $\nu(X)/N$, where the conditional variance $\nu(X)$ is bounded, i.e., there exists $C_{\nu,1} > 0$ such that $\nu(x) \leq C_{\nu,1}$ for all $x \in \mathbb{R}$.*

Let $\rho_M = \frac{1}{M}\sum_{i=1}^{M} h(g(X_i))$ be the nested Monte Carlo estimator with the true function g. The MSE of the standard procedure can be decomposed into two terms.

$$
\begin{aligned}
&\mathbb{E}\left[(\hat{\rho}_{M,N}-\rho)^2\right] \\
&\leq 2\mathbb{E}\left[(\hat{\rho}_{M,N}-\rho_M)^2\right]+2\mathbb{E}\left[(\rho_M-\rho)^2\right] \\
&= 2\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M}h\left(\hat{g}_N(X_i)\right)-\frac{1}{M}\sum_{i=1}^{M}h\left(g(X_i)\right)\right)^2\right] \\
&\quad +2\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M}h\left(g(X_i)\right)-\mathbb{E}\left[h(g(X))\right]\right)^2\right] \\
&= 2\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M}h\left(\hat{g}_N(X_i)\right)-h\left(g(X_i)\right)\right)^2\right]+\frac{2}{M}\mathrm{Var}(h(g(X))) \\
&= 2\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M}h\left(\hat{g}_N(X_i)\right)-h\left(g(X_i)\right)\right)^2\right]+\mathcal{O}(M^{-1}),
\end{aligned}
\tag{2.3}
$$

where 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.

A Smooth Function h

Assumption 3 *The function h has bounded first and second order derivative, i.e., there exists $C_1 > 0$ such that $|h'(x)| \leq C_1$ for all $x \in \mathbb{R}$, and there exists $C_2 > 0$ such that $|h''(x)| \leq C_2$ for all $x \in \mathbb{R}$.*

Assumption 3 is similar to the smoothness assumption in **?**. Since h is a smooth function, Taylor expansion can be applied to the first term in Equation 2.3.

$$
\begin{aligned}
&\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h\left(\hat{g}_N(X_i)\right) - h\left(g(X_i)\right)\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h'\left(g(X_i)\right)\left(\hat{g}_N(X_i) - g(X_i)\right) + \frac{1}{2M}\sum_{i=1}^{M} h''\left(z_i\right)\left(\hat{g}_N(X_i) - g(X_i)\right)^2\right)^2\right] \\
&\leq 2\,\underbrace{\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h'\left(g(X_i)\right)\left(\hat{g}_N(X_i) - g(X_i)\right)\right)^2\right]}_{S_1} \\
&\quad + 2\,\underbrace{\mathbb{E}\left[\left(\frac{1}{2M}\sum_{i=1}^{M} h''\left(z_i\right)\left(\hat{g}_N(X_i) - g(X_i)\right)^2\right)^2\right]}_{S_2}
\end{aligned}
\tag{2.4}
$$

where the last inequality is due to $2ab \leq a^2 + b^2$ for any $a, b \in \mathbb{R}$. For different methods of nested estimation, each of the two terms on the right-hand side of Equation 2.4 can be analyzed separately. We start with the first term S_1.

$$
\begin{aligned}
S_1 &= \mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h'\left(g(X_i)\right)\left(\hat{g}_N(X_i) - g(X_i)\right)\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h'\left(g(X_i)\right)\bar{Z}_N(X_i)\right)^2\right] \\
&\leq C_1^2\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} \bar{Z}_N(X_i)\right)^2\right] \\
&= C_1^2\mathbb{E}\left[\frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} \bar{Z}_N(X_i)\bar{Z}_N(X_j)\right] \\
&= C_1^2\mathbb{E}\left[\frac{1}{M^2}\sum_{i=1}^{M} \bar{Z}_N^2(X_i) + \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j\neq i}^{M} \bar{Z}_N(X_i)\bar{Z}_N(X_j)\right] \\
&\leq \frac{C_1^2 C_{\nu,1}}{MN} = \mathcal{O}(M^{-1}N^{-1}) \qquad (2.5)
\end{aligned}
$$

where the last inequality in Equation 2.5 is due to Assumption 2, independence of X_i and X_j for $i \neq j$, and the fact that $\mathbb{E}\left[\bar{Z}_N(X)\right] = 0$. 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* $x \in \mathbb{R}$.

$$
\begin{aligned}
S_2 &= \mathbb{E}\left[\left(\frac{1}{2M}\sum_{i=1}^{M} h''(z_i)\left(\hat{g}^{\mathrm{SL}}_{M,N}(X_i) - g(X_i)\right)^2\right)^2\right] \\
&= \mathbb{E}\left[\left(\frac{1}{2M}\sum_{i=1}^{M} h''(z_i)\,\bar{Z}^2_N(X_i)\right)^2\right] \\
&\leq C_2^2\mathbb{E}\left[\left(\frac{1}{2M}\sum_{i=1}^{M} \bar{Z}^2_N(X_i)\right)^2\right] \\
&= C_2^2\mathbb{E}\left[\frac{1}{2M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} \bar{Z}^2_N(X_i)\bar{Z}^2_N(X_j)\right] \\
&= C_2^2\mathbb{E}\left[\frac{1}{2M^2}\sum_{i=1}^{M} \bar{Z}^4_N(X_i) + \frac{1}{2M^2}\sum_{i=1}^{M}\sum_{j\neq i}^{M} \bar{Z}^2_N(X_i)\bar{Z}^2_N(X_j)\right] \\
&\leq C_2^2\left(\frac{C_{\nu,2}M}{2M^2N^2} + \frac{C^2_{\nu,1}M(M-1)}{2M^2N^2}\right) = \mathcal{O}(N^{-2})
\end{aligned} \tag{2.6}
$$

where the second inequality is due to Assumption 3, and the last inequality follows from Assumption 4 and the fact that $\hat{g}_N(X)$ is a standard Monte Carlo estimator of $g(X)$. Combining Equation 2.3, 2.4, 2.5, and 2.6, we have

$$
\mathbb{E}\left[(\hat{\rho}_{M,N} - \rho)^2\right] = \mathcal{O}(M^{-1}) + \mathcal{O}(N^{-2}) \tag{2.7}
$$

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 **?**. As shown in Section 2.3.1, the convergence in MSE automatically implies the convergence in probabilistic order.

Theorem 2 *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 2 is completely different from the one used in **?**. The analysis in **?** 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 the

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 1, the convergence in MSE automatically implies the convergence in probabilistic order.

Corollary 1 *Let h be a smooth function. The absolute error 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 **?**, the authors assume the knowing of the corresponding VaR. Hence, the analysis of CVaR is not complete. Instead, the quantity that is being analyzed is instead 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 **?**. For the Lipschitz continuous case, the first term in Equation 2.3 is analyzed differently.

$$\begin{aligned}
\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M} h\left(\hat{g}_N(X_i)\right) - h\left(g(X_i)\right)\right)^2\right] &\leq \mathbb{E}\left[\left(h\left(\hat{g}_N(X)\right) - h\left(g(X)\right)\right)^2\right] \\
&\leq K^2\mathbb{E}\left[\left(\hat{g}_N(X) - g(X)\right)^2\right] \\
&= K^2\mathbb{E}\left[\left(\bar{Z}_N(X)\right)^2\right] = \mathcal{O}(N^{-1}) \qquad (2.8)
\end{aligned}$$

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

$$\mathbb{E}\left[(\hat{\rho}_{M,N} - \rho)^2\right] = \mathcal{O}(M^{-1}) + \mathcal{O}(N^{-1}) \tag{2.9}$$

Setting $M = \mathcal{O}(\Gamma^{1/2})$ and $N = \mathcal{O}(\Gamma^{1/2})$, we provide a looser bound than the one obtained for hockey-stick h.

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

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

Immediately from Theorem 1, the convergence in MSE automatically implies the convergence in probabilistic order.

Corollary 2 *Let h be a Lipschitz continuous function. The absolute error of the standard nested simulation procedure converges in probabilistic order $\Gamma^{-1/4}$, 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 **?**, 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 **?** by providing the 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 **?**.

A kNN regression is a non-parametric method 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 independent and identically distributed (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 the estimate of $Y|X_i$ with an inner simulation budget of n. Let $f_{XY}(x, y)$ be the joint density of random vector (X, Y), and $f_X(x) = \int f_{XY}(x, y)dy$ be the 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 model parameter k estimates the risk measure $\rho = \mathbb{E}[h(g(X))]$ with

$$\begin{aligned} \hat{\rho}^{\text{kNN,Test}}_{M,n,M'} &= \frac{1}{M'} \sum_{i=1}^{M'} h(\hat{g}^{\text{kNN}}_{M,n}(\tilde{X}_i)); \quad \tilde{X}_i \sim F_X, \\ \hat{g}^{\text{kNN}}_{M,n}(x) &= \frac{\frac{1}{MR_k^d} \sum_{i=1}^{M} K\left(\frac{x - X_i}{R_k}\right) \bar{Y}_{n,i}}{\frac{1}{MR_k^d} \sum_{i=1}^{M} K\left(\frac{x - X_i}{R_k}\right)} \end{aligned}$$

where R_k is the Euclidean distance between X_i and its k^{th} nearest neighbor, and $K(\cdot)$ is the kernel function satisfying

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

where $\|\cdot\|$ is the Euclidean norm. 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 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

$$\int \|u\|^2 |K(u)| du < \infty,$$

$$\int \nu_\alpha K(u) du = 0, \quad \text{for } \alpha = 1, \ldots, d.$$

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

The following lemma from **?** characterizes the pointwise convergence of the kNN estimator in terms of its bias and variance at point x.

Lemma 1 *Assume the random vector* (X, Y) *and the 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 Taylor expansion.

$$\begin{aligned} &\mathbb{E}\left[h(\hat{g}_{M,n}^{\text{kNN}}(X)) - h(g(X)) \right] \\ &= \mathbb{E}\left[h'(g(X)) \left(\hat{g}_{M,n}^{\text{kNN}}(X) - g(X) \right) + \frac{1}{2} h''(z) \left(\hat{g}_{M,n}^{\text{kNN}}(X) - g(X) \right)^2 \right] \end{aligned} \tag{2.10}$$

where the bias is decomposed into two terms, and z is between $\hat{g}_{M,n}^{\text{kNN}}(X)$ and $g(X)$. The first term in Equation 2.10 is

$$\begin{aligned} &\mathbb{E}\left[h'(g(X)) \left(\hat{g}_{M,n}^{\text{kNN}}(X) - g(X) \right) \right] \\ &= \mathbb{E}\left[h'(g(X)) \mathbb{E}\left[\hat{g}_{M,n}^{\text{kNN}}(X) - g(X) | X \right] \right] \\ &= \int_\Omega h'(g(x)) B_M(x) f_X(x) dx \\ &= \int_\Omega h'(g(x)) \left(A_{\text{kNN}}(x) \left(\frac{k}{M} \right)^{2/d} (1 + o_x(1)) + B_{\text{kNN}} \cdot \left(\frac{1}{k} + o_x\left(\frac{1}{k} \right) \right) \right) f_X(x) dx \\ &= \left(\frac{k}{M} \right)^{2/d} \mathbb{E}\left[h'(g(X)) A_{\text{kNN}}(X) \right] (1 + o(1)) + \frac{B_{\text{kNN}}}{k} \mathbb{E}\left[h'(g(X)) \right] (1 + o(1)). \end{aligned} \tag{2.11}$$

The second term in Equation 2.10 is

$$
\begin{aligned}
&\mathbb{E}\left[h''(z)\left(\hat{g}_{M,n}^{\mathrm{kNN}}(X)-g(X)\right)^2\right] \\
&= \int_\Omega h''(g(z))\left(B_M^2(X)+V_M(x)\right) f_X(x)dx \\
&\leq C_2 \int_\Omega \left(\frac{C_{\mathrm{kNN}} \cdot \mathrm{Var}[Y|X=x]}{k} \int K^2(u)du + o\left(\frac{1}{k}\right)\right) f_X(x)dx \\
&+ C_2 \int_\Omega \left(A_{\mathrm{kNN}}(x)\left(\frac{k}{M}\right)^{2/d}(1+o_x(1)) + B_{\mathrm{kNN}} \cdot \left(\frac{1}{k}+o_x\left(\frac{1}{k}\right)\right)\right)^2 f_X(x)dx \qquad (2.12)
\end{aligned}
$$

where the second term in Equation 2.12 is of higher order than Equation 2.11, and it is negligible since $kM^{-1} \to 0$ as $M \to \infty$ and $k \to \infty$, The first term in Equation 2.12 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 3 *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,*

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

Definition 3 is obtained by matching the orders of the bias terms in Equation 2.11 and 2.12. 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 **?** 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 2.2, where the orders of convergence are presented in the order of the total simulation budget Γ.

Estimator for $g(X)$	**Smooth h**	**Lipschitz / hockey-stick h**	**Indicator h**	**VaR**	**CVaR**
Standard Monte Carlo	$\mathcal{O}(\Gamma^{-2/3})$	$\mathcal{O}(\Gamma^{-1/2})$ / $\mathcal{O}(\Gamma^{-2/3})$	$\mathcal{O}(\Gamma^{-2/3})$	$\mathcal{O}(\Gamma^{-2/3})$	$\times$
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 2.2: 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

? and **?** 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 **?** further requires assumptions on $f_{Y,\tilde{Z}_N}(y, z_N)$, the 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,

$$\sup_N \int_{-\infty}^{\infty} |z|^r p_{i,N}(z) dz < \infty \tag{2.13}$$

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 **?** only analyze the nested expectation case with smooth and Lipschitz continuous function h, the regression-based nested simulation procedure does not require Assumption 7 and 8 for the convergence analysis. In our analysis of the standard nested simulation procedure for smooth h in the nested expectation form, we make additional assumptions on the fourth moment of the inner simulation noise, i.e., Assumption 3, so the second-order term in the Taylor expansion can be bounded.

2.4.3 Assumptions for the Multi-level Monte Carlo Procedure

In addition to Assumption 8, the multi-level Monte Carlo 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 **?** show the asymptotic convergence of the multi-level Monte Carlo 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 from 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 $\hat{g}_N^{\mathrm{LR}}(X_i)$ to be an unbiased and strongly consistent estimator of $g(X_i)$, 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, **?** 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 $\bar{Z}_N(X_i)$. In essence, this fixed variance design treats the $(X_i, \bar{Y}_{N,i})$ as independent and identically distributed observations of (X, Y), and the kernel-based nested simulation procedure is analyzed as a nonparametric 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 $\hat{g}_{M,N}^{\mathrm{KS}}(X)$, the kernel smoothing estimator of $g(X)$.

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 $\hat{g}_{M,N}^{\mathrm{KS}}(X)$.

2.4.6 Assumptions for the KRR-Based Procedure

For the KRR-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, 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 KRR-based nested simulation procedure is shown in terms of the absolute error rather than the standard metric of MSE, which is due to the complexity of the KRR method itself.

2.5 Finite-Sample Experiments

The theoretical framework has allowed the 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. 6 nested simulation procedures are considered, namely the standard nested simulation procedure, the multi-level Monte Carlo procedure, the regression-based, kernel smoothing-based, likelihood ratio-based, and kernel ridge regression-based nested simulation procedures. For the standard nested simulation procedure, the bootstrap-based budget allocation strategy from **?** is implemented to estimate the optimal values of outer and inner simulation budgets. For the regression proxy, the Laguerre polynomials up to degree 3 are used as the basis functions. The kNN proxy is implemented for the kernel smoothing-based procedure, and a cross-validation with grid search is used to find the optimal number of

neighbors. The KRR proxy is implemented with a Matérn kernel, and the smoothness, the length scale, and the regularization hyperparameters are found by a cross-validation with Bayesian search (**?**). The procedures are compared for 5 types of risk measures, namely a quadratic tracking error, a mean excess loss over threshold u, a probability of a large loss over threshold u, th VaR, and the CVaR, where the threshold u is set to be the 90% VaR. The procedures are compared for different portfolios.

- Portfolio 1 considers d assets. The portfolio contains 3 European call options written on each asset with strikes 90, 100, and 110, respectively.
- Portfolio 2 considers d assets. The portfolio contains 3 geometric Asian options written on each asset with strikes 90, 100, and 110, respectively.
- Portfolio 3 considers d assets. The 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. The 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. The 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., $d = 1, 2, 5, 10, 20$, while we only consider $d = 1$ for portfolio 5. 525 experimental settings that arise from an exhaustive combination of the above simulation procedures, risk measures, portfolios, and asset dimensions are generated. For each of the scenarios, 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. We start by examining the empirical convergence results for the most basic case, i.e., the quadratic tracking error risk measure for European call options 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 replications. The MSEs of the nested simulation procedures are fitted with a regression line, and the slope of the fitted line is reported as

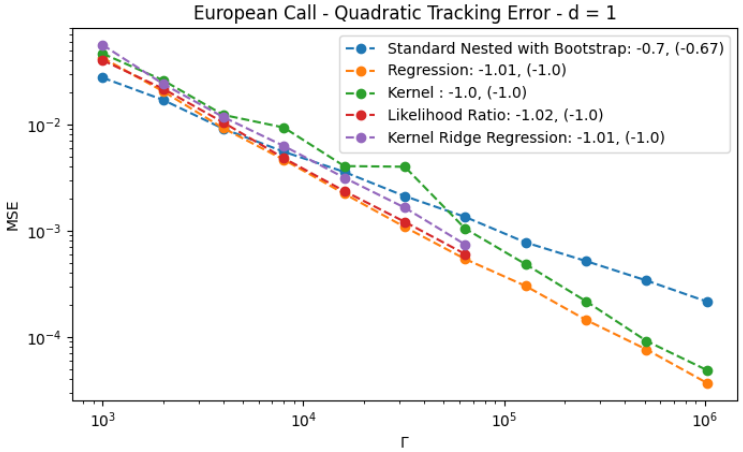


Figure 2.1: Empirical convergence of nested simulation procedures for quadratic tracking error on Portfolio 1 with $d = 1$

the first number in the legend. The second number in the legend is the corresponding asymptotic rate of convergence obtained from the theoretical analysis. The slope of the fitted line can be regarded as the empirical rate of convergence of the corresponding procedure. By comparing the empirical rates of convergence with the asymptotic rates, we can observe that the empirical rates of convergence of the standard, the kernel smoothing-based, the regression-based, the likelihood ratio-based, and the KRR-based procedures all closely match their asymptotic rates. Due to the computational complexity of the likelihood ratio-based and the KRR-based procedures, their MSEs are not reported for Γ larger than $16,000$ and $64,000$, respectively. Due to the difficulty of fixing Γ for the multi-level Monte Carlo procedure, the empirical rate of convergence of the multi-level Monte Carlo procedure is not reported here, but a detailed analysis of the empirical convergence of the multi-level Monte Carlo procedure is provided in Section 2.5.7. In the following sections, we examine the empirical convergence of the nested simulation procedures in a similar manner as in Figure 2.1. With a more detailed analysis of different risk measures, option types, and asset dimensions, we aim to provide a comprehensive understanding 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, the empirical convergence behavior of the nested simulation procedures could be sensitive to the asset dimension. In this section, we examine the empirical convergence of the nested simulation procedures for different asset dimensions.

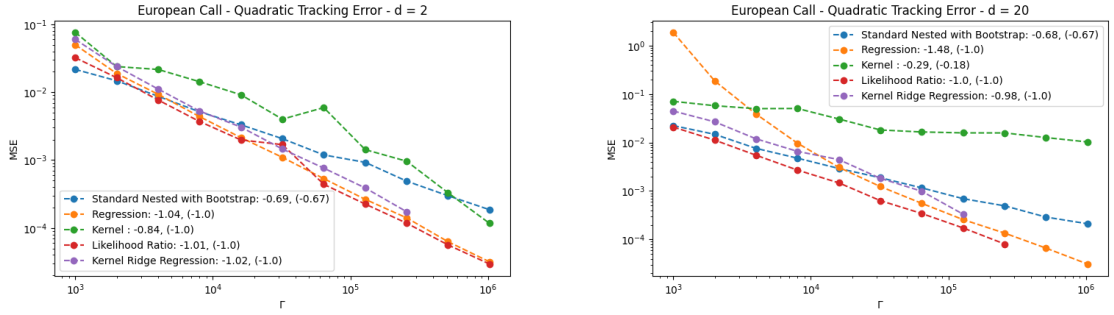


Figure 2.2: Empirical convergence of nested simulation procedures for quadratic tracking error on Portfolio 1 with different asset dimensions

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 KRR-based, and the likelihood ratio-based procedures closely match their asymptotic rates for both asset dimensions, and they are independent of the asset dimension. On the other hand, the empirical rates of convergence of the kernel smoothing-based and regression-based procedures are higher than their asymptotic rates for both asset dimensions. They are sensitive to the asset dimension but in different ways. The empirical rate of convergence of the kernel smoothing-based procedure decreases as the asset dimension increases. This phenomenon is consistent with the theoretical analysis in Section 2.3, where the asymptotic rate of convergence of the kernel smoothing-based procedure is shown to be sensitive to the asset dimension. However, the empirical rate is still higher than its asymptotic rate for $d = 2$ and $d = 20$. The empirical rates of convergence of the regression-based procedure is higher than its asymptotic rates for both asset dimensions, and the empirical rate is higher for $d = 20$ than for $d = 2$.

2.5.2 Empirical Convergence of Parametric Regression Procedures

For both the regression-based and the kernel smoothing-based procedures, the reason for the higher empirical rates can be explained by having poor proxy estimators for smaller simulation budgets. For lower simulation budgets, the proxy estimators of the true inner simulation are poor, and the MSEs of the regression-based and kernel smoothing-based procedures are dominated by the model error of the proxy estimators. In other words, they have not reached their asymptotic regimes yet. 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 it stabilizes after a certain budget level. In 2.3 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 $d = 20$.

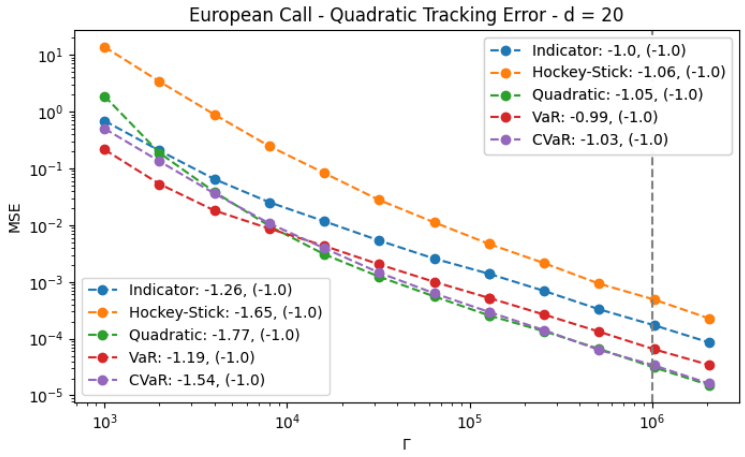


Figure 2.3: Empirical convergence of regression procedure for European call options and $d = 20$

The left part of Figure 2.3 contains the MSEs of the regression procedure for budget sizes that are smaller than $10,000$. Slopes of the fitted lines on the left correspond to the empirical rate of convergence of the regression procedure for budget levels between $1,000$

and $10,000$. To investigate the convergence behavior for higher budget levels, we conduct additional experiments for the European call option with dimension 20. The additional experiments are summarized on the right of Figure 2.3. After reaching a certain budget level, i.e., $\Gamma = 1,000,000$, the empirical rate of convergence for the regression-based nested simulation procedure approaches its asymptotic rate. For nested simulation procedures whose proxy models are biased, the proxy estimators of the true inner simulation are poor, especially for smaller budget sizes. We are able to clearly observe this phenomenon for the regression proxy, and it can be explained by dividing the MSE into proxy bias and simulation variance. For small budget sizes, the improvement of the bias of the regression proxy dominates the improvement of simulation variance. Performing poorly for extremely low simulation budgets, the regression proxy improves significantly. As the simulation budget gets higher, the improvement of the regression proxy becomes negligible compared with that of the simulation variance. The regression proxy ceases to improve after reaching a certain level ($\Gamma = 100{,}000$ in our case), and the improvement of simulation variance dominates.

2.5.3 Empirical Convergence of Kernel Smoothing-Based Procedures

The kernel smoothing proxy is a nonparametric regression procedure. 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 dimension of the asset.

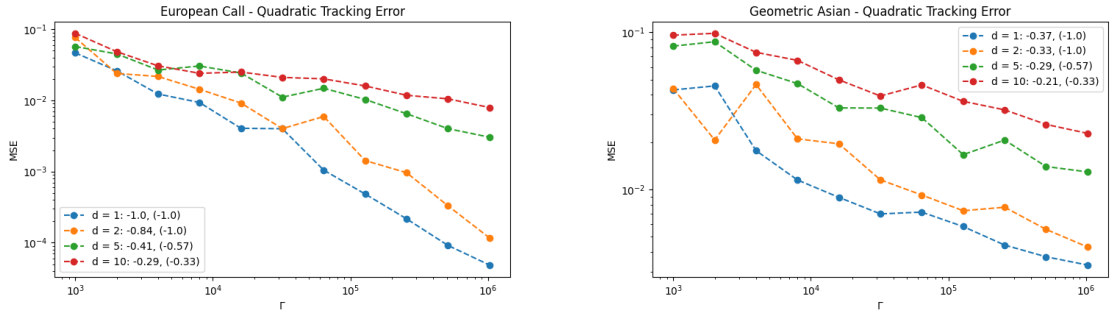


Figure 2.4: Empirical convergence of kernel smoothing procedure for different values of d

In Figure 2.4, the empirical convergence of the kernel smoothing-based nested simulation procedure for quadratic tracking error is illustrated for different asset dimensions, i.e., $d = 1, 2, 5, 10$. The observation 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 significantly 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 2.4 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 nonparametric regression procedure, is highly dependent on the cross-validation of the proxy hyperparameters. For the kernel smoothing-based procedure, the kNN proxy is implemented. Hence, the proxy hyperparameter 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 $1,000$ replications. Therefore, a poor selection of the optimal value of k can lead to a poor proxy estimator, and the MSE of the kernel smoothing-based procedure will be dominated by the model error of the proxy estimator.

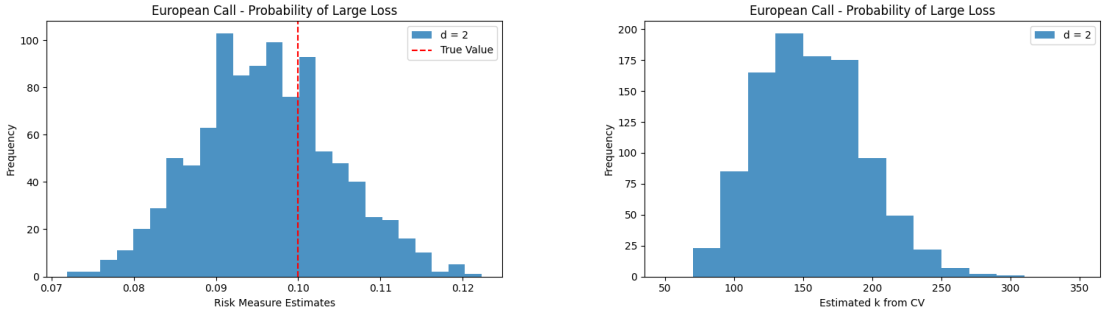


Figure 2.5: Cross-validation for the kernel smoothing-based procedure with $\Gamma = 100,000$

In Figure 2.5, 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 the two replications where $k = 80$ is selected as the optimal value of k, the estimated risk measures for the kernel smoothing-based procedure are 0.1187 and 0.1189, which are significantly 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 dependent on the cross-validation of the proxy hyperparameters.

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 is the most basic option type. To examine the sensitivity of the empirical convergence behavior to the option type, we consider path-dependent options, namely geometric Asian options and barrier options.

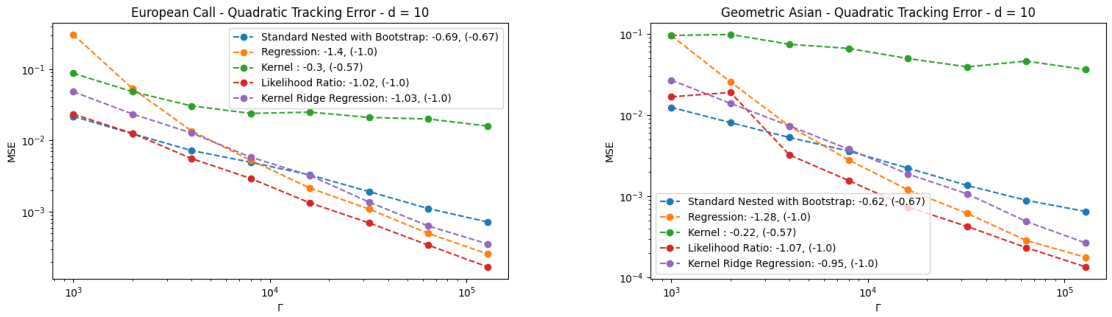


Figure 2.6: Empirical convergence of nested simulation procedures for quadratic tracking error on different portfolios with $d = 20$

In Figure 2.6, 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 KRR-based nested simulation procedures closely ensemble their asymptotic rates for path-dependent options. The empirical rate of convergence of the regression-based and kernel smoothing-based procedures is much higher than their asymptotic rates for barrier options, with reasons explained in Section 2.5.2.

In **?**, the authors propose a numerical example where the payoff has a W-shape with respect to the underlying asset price. We have incorporated this example as Portfolio 5.

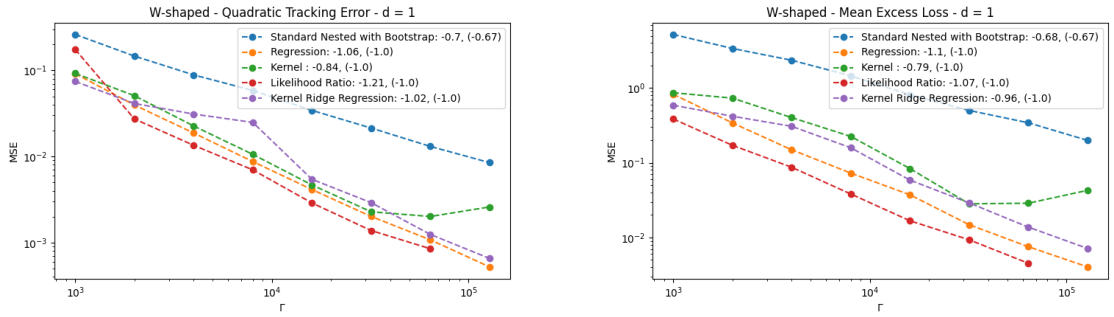


Figure 2.7: Empirical convergence of nested simulation procedures for a W-shaped payoff

In Figure 2.7, the empirical convergence of the nested simulation procedures for quadratic tracking error and 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 KRR-based procedures do not decrease monotonically as the simulation budget increases, as observed and analyzed in Section 2.5.3.

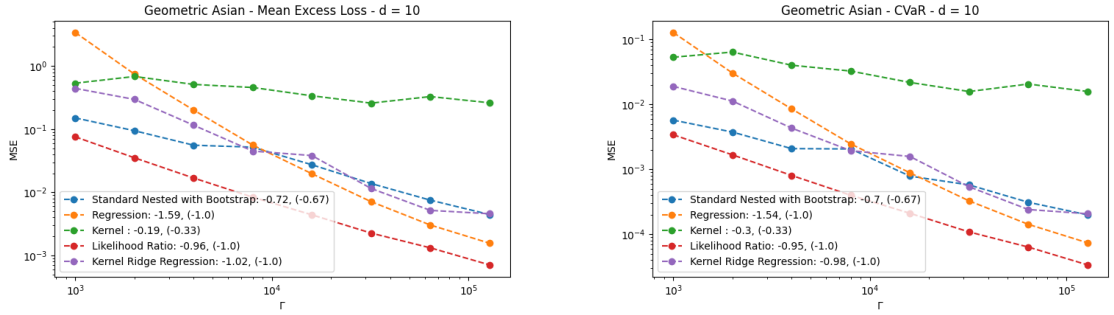


Figure 2.8: Empirical convergence of nested simulation procedures for different risk measures on Portfolio 1 with $d = 20$

Figure 2.8 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 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 budget level, the rate of decrease stabilizes to match its asymptotic rate of convergence.

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 different levels of VaR and CVaR, i.e., 80%, 90%, 95%, 99%, and 99.6%.

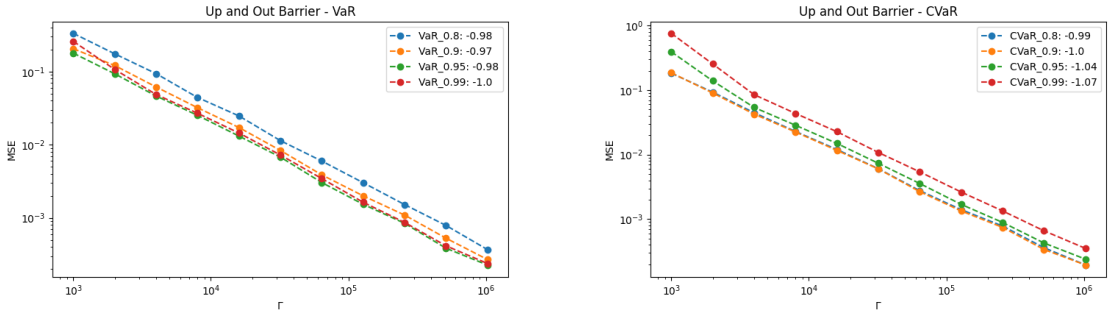


Figure 2.9: Empirical convergence of regression-based procedures for different levels of VaR and CVaR for Up and Out Barrier Call Options

In Figure 2.9, 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 independent of the level of VaR and CVaR.

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, i.e., a Heston model.

In Figure 2.10, 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 independent of the asset model.

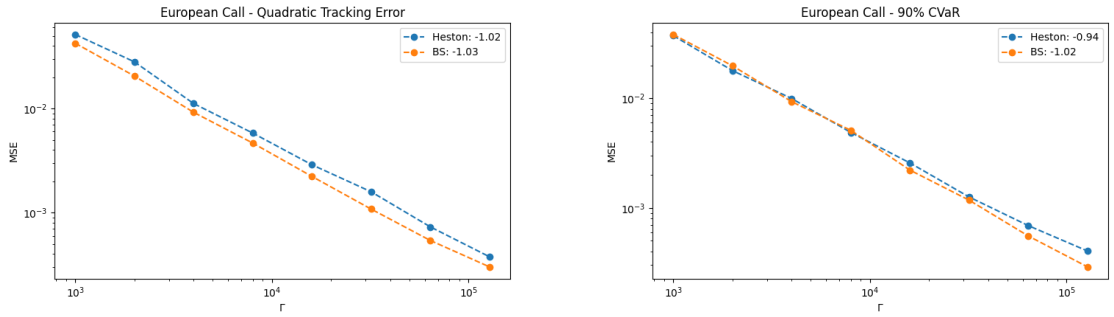


Figure 2.10: Empirical convergence of regression-based nested simulation procedures for different asset models

2.5.7 Empirical Convergence of Multi-level Monte Carlo

The multi-level Monte Carlo procedure is a variance reduction technique that is designed to reduce the computational cost of nested simulation procedures. The theoretical analysis in **?** shows that the multi-level Monte Carlo 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.

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 2.3: MSEs of the multi-level Monte Carlo procedure for different levels

Instead of using a figure, we provide a summary of the empirical convergence results of the multi-level Monte Carlo procedure with a decomposition table of the MSEs in Table 2.3. The MSEs of the multi-level Monte Carlo 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 multi-level Monte Carlo procedure is designed with a fixed number of outer simulation paths, the

benefit of the multi-level Monte Carlo 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 procedure is lower than that of the multi-level Monte Carlo procedure with a fixed number of outer simulation paths.

2.6 Computational Complexity

Compared to the standard procedure, proxy-based nested simulation procedures often have faster empirical rates of convergence. The faster convergence benefits from the pooling of inner samples through the proxy models. However, pooling itself comes with 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 2.4: Additional computational costs of nested simulation procedures aside from simulation

Table 2.4 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 2.4, as all methods 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 thus can be written in terms of M only. The training cost includes the cost of fitting the proxy model and the cost of cross-validation for nonparametric regression proxies. For the standard procedure, the training cost is 0 due to the absence of proxy models. 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. In practice, p is usually much smaller than M and does not grow with the simulation budget Γ. However, it is dependent on the complexity of the payoff function. 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 matrix inversion, see **?**. The matrix inversion has complexity $\mathcal{O}(p^3)$ using Gauss-Jordan elimination, but in practice, the complexity can be reduced to $\mathcal{O}(p^{2.807})$ by **?** and $\mathcal{O}(p^{2.376})$ by **?**. 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 proxy is implemented. In practice, the K-D tree algorithm (**?**) is often implemented for efficient nearest neighbor search. During training, a tree is constructed to store the distance, which has complexity $\mathcal{O}(M\log(M))$. 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 complexity of $\mathcal{O}(M^2)$ and $\mathcal{O}(M^2\log(M))$, respectively. A practical implementation of block sort is provided in **?**. Hence, an efficient implementation of the kNN proxy 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 KRR-based procedure, the main training cost comes from inverting an $M \times M$ kernel matrix (**?**), which has complexity $\mathcal{O}(M^3)$ using 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 KRR-based procedure is $\mathcal{O}(M^2)$, which comes from the multiplication of the kernel matrix by the estimated coefficients. Among the proxy-based procedures, regression is the most efficient as p is usually much smaller than M. Kernel smoothing and KRR are kernel-based proxies, and they are more expensive due to distance calculation and cross-validation of hyperparameters. The kNN proxy has only 1 hyperparameter k, while the KRR proxy requires 3 hyperparameters, namely the smoothness hyperparameter ν, scale hyperparameter ℓ, and the 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 KRR requires Bayesian optimization (**?**) 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 proxy-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.

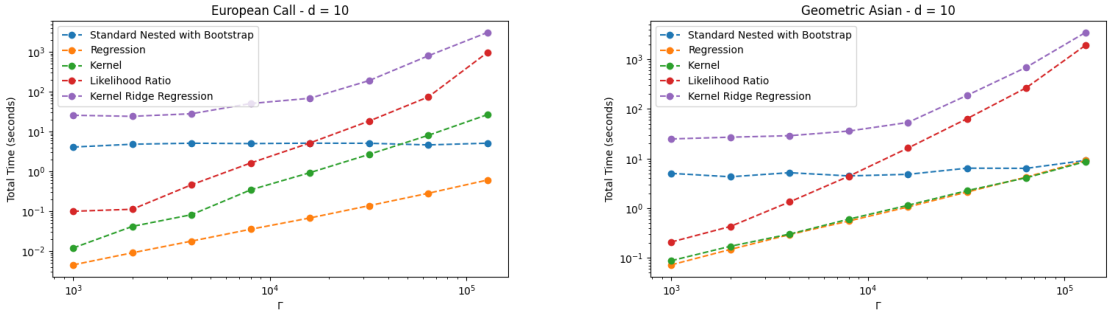


Figure 2.11: Total computational cost for different procedures with $d = 10$

In our finite-sample experiments, we have observed that the actual computational cost to be significantly different from the theoretical complexity, as not only the order of complexity but also the constant factors can affect the computational cost. In Figure 2.11, the total 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, 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 32GB 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 proxy-based procedures. Since the regression-based procedure has a higher empirical rate of convergence, it is preferred over the kNN. The likelihood ratio-based and KRR-based procedures are the most computationally expensive among all procedures. They are 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 KRR-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 2.12 illustrates the computational time of implementing different nested simulation procedures for the portfolio of geometric Asian options with $d = 10$, excluding the simulation time. The simulation time is omitted as it is the same for all procedures. The remainder can be decomposed into two parts: hyperparameter tuning and model

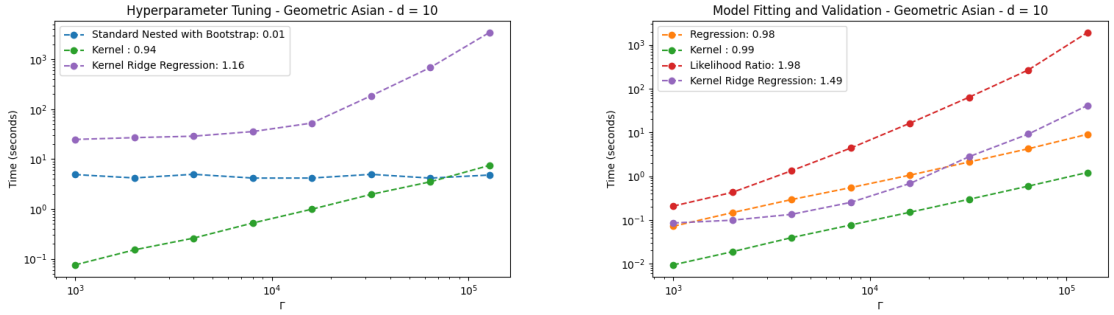


Figure 2.12: Computational cost for implementing nested simulation procedures with $d = 10$, excluding simulation time

implementation. The hyperparameter tuning cost is the cost of estimating the optimal hyperparameters for the proxy models, and the model implementation cost is the cost of fitting the proxy models and generating predictions from the trained proxies. For each procedure, each point in Figure 2.12 represents the average computational time for a given simulation budget Γ. A regression model is fitted to each model respectively, and its slope is reported 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 **?**. This cost does not grow with the simulation budget Γ, as its slope is close to 0. The regression-based procedure does not require 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 the its algorithmic complexity in Table 2.4. The kernel smoothing-based and KRR-based procedure requires cross-validation to estimate the optimal hyperparameters. In our implementaion of kNN, the cross-validation is conducted using a grid search with a search space whose 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 KRR-based procedure requires Bayesian optimization to estimate the optimal hyperparameters, and the cardinality of the search space grows with the simulation budget Γ. The empirical cost of the KRR-based procedure does not resemble its algorithmic complexity in Table 2.4. The efficient implementation of KRR is an active area of research, and the computational cost of KRR can be reduced by using a low-rank approximation of the kernel matrix, e.g., the Nyström method (**?**). Our observation for KRR implementation time is in line with the findings in **?**, where the cost of KRR is reported quickly increasing 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 proxy-based procedures in terms of computational time.

2.7 Conclusion

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 proxy model. In this study, we review the literature on nested simulation procedures in financial engineering and establish fair comparisons of different proxy models using the same set of numerical examples. Asymptotic properties of estimators for different nested simulation procedures are influenced by their corresponding proxy models. To show an asymptotic convergence result, a more complex proxy model 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 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 computationally expensive to compute and 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 KRR, requires cross-validation, and its empirical performance depends heavily the choice of hyperparameters. A kNN-based procedure is sensitive to the asset dimension and the problem complexity. A KRR-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 option portfolios.

Chapter 3

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

3.1 Introduction

Deep neural networks (**??**) have attracted attentions of researchers and practitioners due to their success in solving real-world supervised learning tasks such as AlphaGo (**?**) and ChatGPT (**?**). Since the first artificial neural network model (**?**) and the first algorithm for training a perceptron (**?**), especially after the introduction of backpropagation (**?**) and the growth of high-performance computing, the field of artificial neural network and deep learning in general has grown rapidly. Two specialized neural network architectures that are relevant to our study are recurrent neural networks (RNNs) (**??**) and long-short-term memory (LSTM) (**??**), as we need to train metamodels[1] that take sequential observations as input. Despite their success, deep neural network models are often criticized for their lack of transparency and interpretability, which hinders their adoption in financial and actuarial applications. Enormous research efforts are spent to test and improve the robustness of deep neural network models with carefully designed noise injection methods. **?** show that injecting synthetic noise before and after hidden unit activations during training improves

[1] In the field of actuarial science, a metamodel is commonly referred to as a proxy model.

the performance of autoencoders. **?** improve learning for deep neural networks 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, **?** show that adding synthetic label noise to the convolutional neural network (CNN) can improve its ability to capture global features. **?** quantify the error tolerance by injecting synthetic label noise with a custom Boltzmann machine hardware. **?** find that neural networks are vulnerable to adversarial examples, and **?** design an efficient method to generate such noisy examples to exploit the vulnerability to adversarial perturbations. **?** design targeted attacks to training labels to test the robustness of neural networks. Instead of using synthetic noise, **?** 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 deep neural networks in financial and actuarial applications has been received by regulators with some skepticism.

The contributions of our study are two-fold:

1. We study what deep neural networks 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 deep neural networks 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: (1) to identify a set of tail scenarios, and (2) 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 the second stage, while the second procedure uses metamodel predictions to estimate risk measures directly. Our numerical results show that deep neural network metamodels can identify the tail scenarios accurately and so the proposed procedures can estimate tail risk measures with similar accuracy while, at the same time, using less simulation budget.

We are curious about fundamental questions like "What do deep neural networks learn from noisy data?" and "How well do neural networks learn from noisy data?". Data-driven answers to these questions prevail in the existing literature. In supervised learning, deep neural networks are believed to learn from the given data about the feature-label relationship to predict new labels for unseen features. Cross validation using 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. But 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 (splitting it into multiple subsets), we propose using stochastic simulation outputs as training labels for deep neural network models. 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.

In nested simulation, a simulation model is used to generate a large number of outer scenarios, and each scenario is then used as an input to another simulation model. Borrowing terminologies from machine learning research, we can view a set of simulated outer scenarios and the estimated hedging errors for those scenarios as the *features* and (noisy) *labels*. One can train supervised learning models using these simulated features and labels. They are then used to replace the time-consuming inner simulations by the trained model. We refer to the trained supervised learning models as *metamodels* of the inner simulation, which is also known as the *surrogate models*. 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 deep neural networks as metamodels is a relatively new development. **?** provide general guidelines for simulation metamodeling with neural networks, **?** use deep neural networks as metamodels of a simulation model for structural reliability analysis, and **?** show that neural network metamodels help achieve higher prediction accuracy that other metamodels in approximating agent-based simulation models. A popular metamodel in nested simulation procedures is stochastic kriging. **?** use stochastic kriging as a metamodel of Monte Carlo simulations to estimate the Conditional Value-at-Risk (CVaR) of a portfolio of derivative securities, and **?** use stochastic kriging for an efficient valuation of large portfolios of variable annuity (VA) contracts. Other studies, such as **?**, **?**, and **?** use regression, kernel smoothing, and the likelihood ratio method, respectively. 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 neural network models in predicting the noiseless labels.

The rest of this article is organized as follows:

3.2 Problem Formulation

In this section we present notations, problem settings, and a simulation model for risk estimation for hedging errors of variable annuities. A main goal of the section is to showcase the complexity of the simulation model, which we use as a data generator to train deep neural network 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: VaR and CVaR

Measuring and monitoring risks, particularly tail risks, are important risk management tasks for financial institutions like banks and insurance companies. Two most popular tail risk measures are Value-at-Risk (VaR) and Conditional Value-at-Risk (CVaR) (**?**). Other

names of CVaR include Conditional Tail Expectation (CTE), Tail Value-at-Risk (TailVaR), and Expected Shortfall (ES).

Consider a loss random variable L whose 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 is defined as the α-quantile of L: $\text{VaR}_\alpha = q_\alpha = \inf\{q : \Pr(L \leq q) \geq \alpha\}$. The α-CVaR of L is defined as $\text{CVaR}_\alpha = \frac{1}{1-\alpha}\int_{v=\alpha}^{1} q_v dv$. Tail risk measures like VaR and CVaR are widely used for setting 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 **?**. In Canada, the regulatory capital requirement for VAs is set based on CVaRs as prescribed in **?**.

Let $L_1, L_2, \ldots, L_M$ be M independent and identically distributed (i.i.d.) simulated losses of L and let $L_{(1)} \leq L_{(2)} \leq \ldots \leq L_{(M)}$ be the corresponding 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

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

where 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

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

Two relevant types of guarantees in our studies are:

- **Guaranteed Minimum Maturity Benefit (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.
- **Guaranteed Minimum Withdrawal Benefit (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 GMDB is usually offered. Additional features offered with the GMWB include roll-up, ratchet, and reset (**?**).

For a comprehensive review of other types of VA contracts such as Guaranteed Minimum Death Benefit (GMDB), Guaranteed Minimum Accumulation Benefit (GMAB) and Guaranteed Lifetime Withdrawal Benefit (GLWB), we refer readers to **?**. Next we present a summary of dynamic hedging for VA contracts. We refer readers to **?** 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 $\tau > 0$. 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. Note that the policyholder's (random) time of death also affects the timing of the benefit payout for certain types of VA such as GMDB, 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 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 sub-account value changes according to the growth of the underlying stock and the (gross) management fee is deducted. That is,

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

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., $F_t \eta_n$.

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

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

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

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

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) whose evolution is driven by the stochasticity of S_t. In practice, the simulation model may also incorporate additional complications like 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 GMMB contract is $V_t = e^{-r(T-t)} \cdot (G_T - F_T)^+ - \sum_{s=t+1}^{T} e^{-r(T-s)} F_s \eta_n$. Also, the insurer's time-t liability for a GMWB contract is $V_t = \sum_{s=t+1}^{T} e^{-r(T-s)} [(I_s - F_s)^+ - \eta_n F_s]$.

For example, consider the time-t liability V_t of a GMWB: Suppose that given the stock sample path, e.g., an outer path $S_1, \ldots, S_t$, one can simulate future stock prices $\widetilde{S}_{t+1}, \ldots, \widetilde{S}_T$, e.g., inner sample paths, based on some asset model such as a Black-Scholes model. The tilde symbol ($\sim$) over a quantity denotes its association with the inner simulation. Given the time t state (S_t, F_t, G_t), following **?** the sensitivity of V_t with respect to S_t can be estimated by a pathwise estimator (**?**):

$$\Delta_t(\widetilde{S}_{t+1}, \ldots, \widetilde{S}_T | S_t) = \frac{\partial V_t}{\partial S_t} = \sum_{s=t+1}^{T} e^{-r(T-s)} \left[\mathbf{1}\{\widetilde{I}_s > \widetilde{F}_s\} \cdot \left(\frac{\partial \widetilde{I}_s}{\partial S_t} - \frac{\partial \widetilde{F}_s}{\partial S_t} \right) - \eta_n \frac{\partial \widetilde{F}_s}{\partial S_t} \right], \quad t = 0, \ldots, T-1, \tag{3.1}$$

where $\mathbf{1}\{\cdot\}$ is an indicator function and

$$\frac{\partial \widetilde{F}_s}{\partial S_t} = \mathbf{1}\{\widetilde{I}_{s-1} < \widetilde{F}_{s-1}\} \cdot \left(\frac{\partial \widetilde{F}_{s-1}}{\partial S_t} - \frac{\partial \widetilde{I}_{s-1}}{\partial S_t} \right) \cdot \frac{\widetilde{S}_s}{\widetilde{S}_{s-1}} \cdot (1 - \eta_g),$$
$$\frac{\partial \widetilde{G}_s}{\partial S_t} = \mathbf{1}\{\widetilde{G}_{s-1} < \widetilde{F}_s\} \cdot \frac{\partial \widetilde{F}_s}{\partial S_t} + \mathbf{1}\{\widetilde{G}_{s-1} \geq \widetilde{F}_s\} \cdot \frac{\partial \widetilde{G}_{s-1}}{\partial S_t},$$
$$\frac{\partial \widetilde{I}_s}{\partial S_t} = \gamma \frac{\partial \widetilde{G}_s}{\partial S_t}.$$

The recursion is initialized with $(\widetilde{S}_t, \widetilde{F}_t, \widetilde{G}_t) = (S_t, F_t, G_t)$, $\frac{\partial \widetilde{F}_s}{\partial S_t} = \frac{\widetilde{F}_t}{S_t}$, and $\frac{\partial \widetilde{G}_s}{\partial S_t} = \frac{\partial \widetilde{I}_s}{\partial S_t} = 0$.

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.

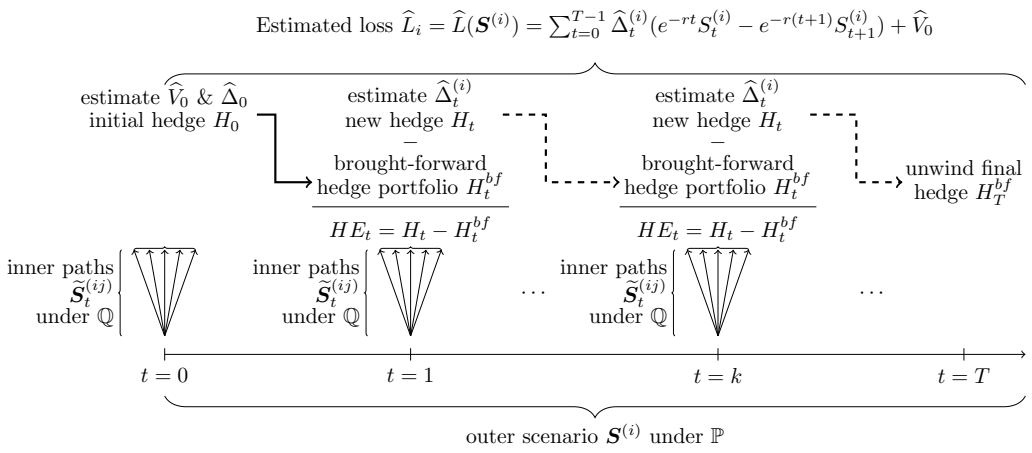


Figure 3.1: Illustration of multi-period nested simulation that estimates the P&L for one outer scenario.

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 in Section 3.3 and Section 3.4 can be trivially adapted to more general hedging strategies.

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:

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

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

$$HE_t = H_t - H_t^{bf}, \quad t = 1, \ldots, T-1. \tag{3.2}$$

The P&L of the VA contract includes the cost of the initial hedge (H_0), the hedging errors (3.2), the unwinding of the hedge at maturity (H_T^{bf}), and the unhedged liability (V_0). Mathematically, the present value of these cash flows is given by

$$L = H_0 + \sum_{t=1}^{T-1} e^{-rt} HE_t - e^{-rT} H_T^{bf} + V_0 = \sum_{t=0}^{T-1} \Delta_t (e^{-rt} S_t - e^{-r(t+1)} S_{t+1}) + V_0, \tag{3.3}$$

where the second equality holds by a telescopic sum simplification of $e^{-rt} B_t$, $t = 0, \ldots, T-1$.

In (3.3), 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. Recall from Section 3.2.2 that the stock sample path, regardless of the inner or outer simulation or a combination of both, determines the evolution of the triplet (S_t, F_t, G_t). Specifically, the outer scenarios $\boldsymbol{S}^{(i)} = (S_1^{(i)}, \ldots, S_T^{(i)})$, $i = 1, \ldots, M$ are generated under $\mathbb{P}$. 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.

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

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

- each inner simulation is time-consuming, as it includes T simulation experiments, one at each time $t = 0, \ldots, T-1$,
- after running inner simulations for M scenarios, we obtain simulated data, that is, 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, that 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 inspired by **?**, we propose a two-stage nested simulation procedure which uses a deep neural network 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 neural network metamodel that

aims to produce a CVaR estimate that's as accurate as that of the standard procedure, but uses fewer computations as than 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 neural network. Refer to the trained model as a metamodel and denote it by $\widehat{L}^{PD}(\boldsymbol{S})$. Denote the predicted losses for the outer scenarios by $\widehat{L}_i^{PD} = \widehat{L}^{PD}(\boldsymbol{S}^{(i)})$, $i = 1, \ldots, M$.

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

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

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

Similar to **?**, 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 competing simulation procedure (or a benchmark). There are two different 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 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 Stage 2 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, then the two-stage procedure is expected to achieve higher 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 this fixed-accuracy design is to investigate whether deep neural network metamodels trained with much noisier labels can identify true tail scenarios with similar 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 $(1-\alpha)M$ largest 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 more true tail scenarios are included in the 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 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 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.

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 $\widehat{\text{CVaR}}_\alpha^{PD} = \frac{1}{(1-\alpha)M} \sum_{i=\alpha M+1}^{M} \widehat{L}_{(i)}^{PD}$.

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 the running the standard procedure in Stage 2, (2) the safety margin ϵ is not needed in the single-stage procedure, and (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 knowing of the entire loss distribution, such as the standard deviation or the squared tracking error. In our numerical experiments, we will compare the single-stage procedure to the two-stage procedure and the standard procedure in estimating the α-CVaR of the hedging losses for GMWB, and we will also examine the single-stage procedure's performance in estimating the standard deviation of the hedging losses.

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 **?**: 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. The risk-free rate is 0.2% per period and the underlying asset S_t is modeled by a regime-switching geometric Brownian motion with parameters specified in Table 2 of **?**.

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 5,000 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.

We compare our two-stage procedure to a standard nested simulation procedure that runs $N = 1{,}000$ 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 research, we split this dataset into three parts: The training, validation, and test sets have 90,000, 5,000, and 5,000 data points (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, $N = 1{,}000$ inner replications are run for all predicted tail scenarios. Stage 2's simulation budget is $\frac{m}{M}$ of the standard procedure's. In short, the two-stage procedure uses $15\% - 30\%$ of the standard procedure's budget for a safety margin between $0\% - 15\%$.

Five metamodels are considered in this experiment: multiple linear regression (MLR), quadratic polynomial regression (QPR) without interaction terms, feed-forward neural network (FNN), recurrent neural network (RNN), and long short-term memory (LSTM) network. MLR and QPR are considered as extensions of regression metamodels in the nested simulation literature. 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 Rectified Linear Unit (ReLU) activation function is used for the fully-connected layers. All neural network metamodels are trained by the Adam optimizer (**?**) with an initial learning rate of 0.001 and an exponential learning rate decay schedule. FNN is trained with a dropout rate of 20%. RNN and LSTM 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 zero mean and unit standard deviation. The architectures and the numbers of trainable parameters are shown in the first two columns of Table 3.1. We see that the three deep neural network metamodels' have orders of magnitudes more trainable parameters than the two regression models. In machine-learning terminologies, the three deep neural network models have much higher *model capacities*.

Model	**Layer size**	**Capacity**	**Training error**	**Test error**	**True error**
MLR	N/A	241	0.707($\pm$0.000)	0.877($\pm$0.001)	0.694($\pm$0.000)
QPR	N/A	481	0.544($\pm$0.000)	0.636($\pm$0.001)	0.531($\pm$0.000)
FNN	$240, 128, 16$	35,009	0.118($\pm$0.001)	0.247($\pm$0.002)	0.113($\pm$0.001)
RNN	$(240, 32), (240, 4), 32$	32,021	0.132($\pm$0.008)	0.137($\pm$0.008)	0.119($\pm$0.008)
LSTM	$(240, 32), (240, 4), 32$	35,729	0.075($\pm$0.004)	0.079($\pm$0.005)	0.063($\pm$0.004)
RNN*[2]	$(240, 32), (240, 4), 32$	32,021	0.109($\pm$0.005)	0.128($\pm$0.005)	0.109($\pm$0.005)

Table 3.1: Architectures and MSEs of metamodels for GMWB inner simulation model.

For each metamodel, 50 independent macro replications are run[3]. The last three columns in Table 3.1 display the average squared errors between the predicted losses and the losses (labels) in different datasets. The half lengths of their 95% confidence interval of these quantities are also reported in parentheses. We first observe that the two regression metamodels' errors are larger than those of three deep neural network models. This is because model capacities of the regression metamodels are too low to learn the complex dynamic hedging simulation model. There are also differences among the three deep neural neetwork models. The FNN is a generic neural network, its test error is larger than the training error, which is a sign of over-fitting and poor generalization. In contrast, RNN and LSTM networks have recurrent strucutures and are designed to capture temporal relationship in the high-dimensional feature. As a result, they have lower training errors, i.e., they fit the data better, and have lower testing error, i.e., they generalize better. Notably, the true errors for the deep neural network metamodels are lower than the test errors.

[2]Results of the well-trained RNN, where the ones suffers from the vanishing gradient issue are removed from the averages.

[3]Each macro replication includes simulating a noisy dataset, separating it into training and test data, fitting a metamodel, and making predictions

Comparing the two metamodels with recurrent structures, A LSTM network overcomes two key drawbacks of a crude RNN.

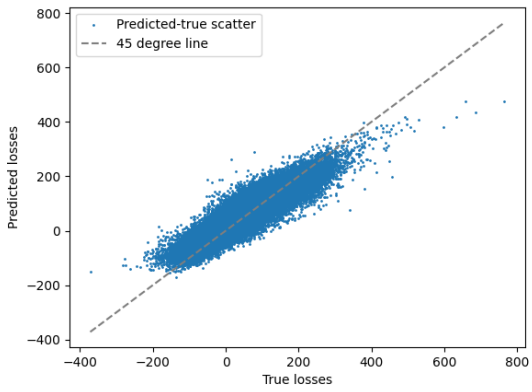
1. It avoids the vanishing gradient problem during training, which makes the LSTM easier to than the RNN. Figure 3.2 shows the quantile-quantile (QQ) plots between the **training** labels and the RNN predictions for two macro replications, where training of the latter is hindered by the vanishing gradient problem[4]. As a result, the half length of the 95% confidence interval of the RNN's training error is much larger than all other metamodels. The last row of Table 3.1 shows the average squared errors of the good RNN metamodels that are 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, but the half lengths of their 95% confidence intervals are larger than those of the FNN. This suggests that the RNN is more sensitive to noise in stochastic gradient descent than the FNN.

2. A LSTM introduces three gates that regulate the flow of information. These gates decide what information should be kept or discarded. This makes a LSTM more capcable of learning long-term dependencies. Since the feature in our dynamic hedging problem is a 240-dimensional time series, a LSTM is more suitable than a crude RNN.

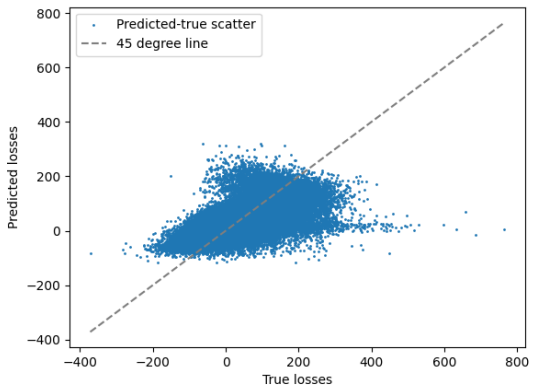
Recall that all three data sets are generated from the same simulation model, but the true data set is less noisy than the training and test data. Part of the training error is due to simulation noise, and this noise is lower in the true error. We observe that the two regression metamodels not only generalize to the test data poorly but also generalize to the true data poorly. In contrast, the deep neural network metamodels generalize better to the true data than to the test data. Figure 3.3 and Figure 3.4 shows the QQ plots between the **true** labels and the predicted losses for the regression metamodels and the neural network metamodels, respectively.

Compared with the MSE table in Table 3.1 that summarizes the overall fit, QQ plots offer a close look at the metamodels' performance on different parts of the loss distribution. Figure 3.3 show that the regression metamodels' predictions are far from the true losses, especially on the tails. Between the two regression metamodels, the QPR metamodel has a slightly better fit for larger losses. Nevertheless, both regression metamodels have poor fits to the true data, and their fit on the tail is particularly undesirable. The poor fit on

[4]A well-trained RNN stops early at around 300 training epochs, while an ill-trained RNN uses up all 1000 training epochs.

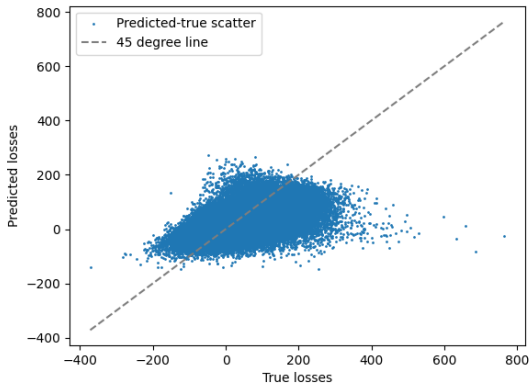


(a) A good RNN metamodel

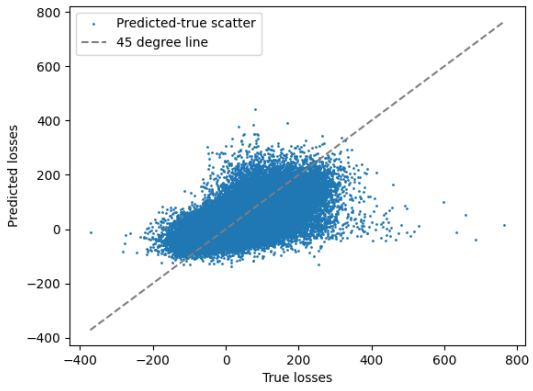


(b) A bad RNN metamodel

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



(a) MLR metamodel

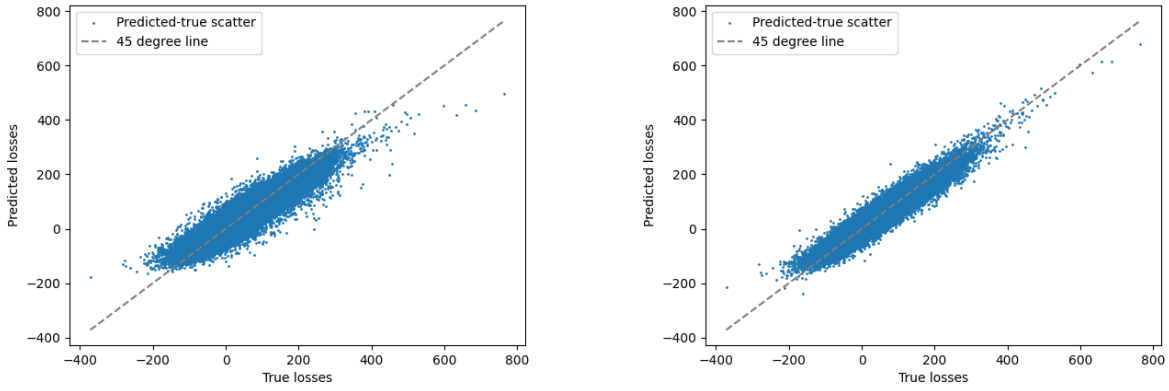


(b) QPR metamodel

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

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. Having 240 features, feature engineering for regression metamodels is not a trivial task and is highly dependent on the simulation model. As a result, the regression metamodels are not flexible enough to capture the complex feature-label relationship in the dynamic hedging simulation model.



(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.

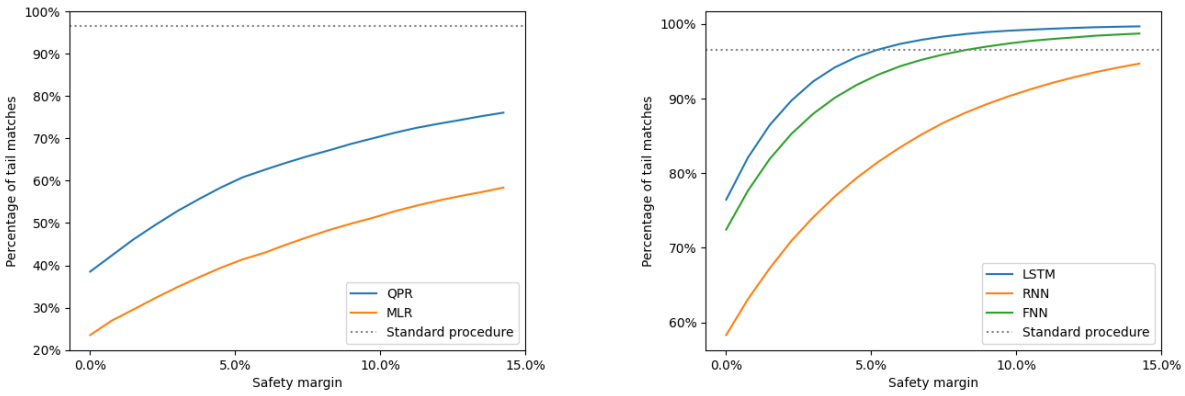
In Figure 3.4, we illustrate the fit of the neural network metamodels. The FNN has a better fit than the regression metamodels, but it still has a poor fit on the tails. We observe that the LSTM metamodels' QQ-plots closely follow that 45-degree line. Again, these metamodels are trained on noisy data, 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 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 deep neural network 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 through adjusting the number of inner replications while keeping the same simulation procedure. As a result, this approach provides a controlled environment to examine the impact of label noise on neural network model. 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 3.1 and the QQ plots provide direct evidence that deep neural network 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.

3.5.1 Two-Stage Procedure

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 3.1 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) Regression metamodels

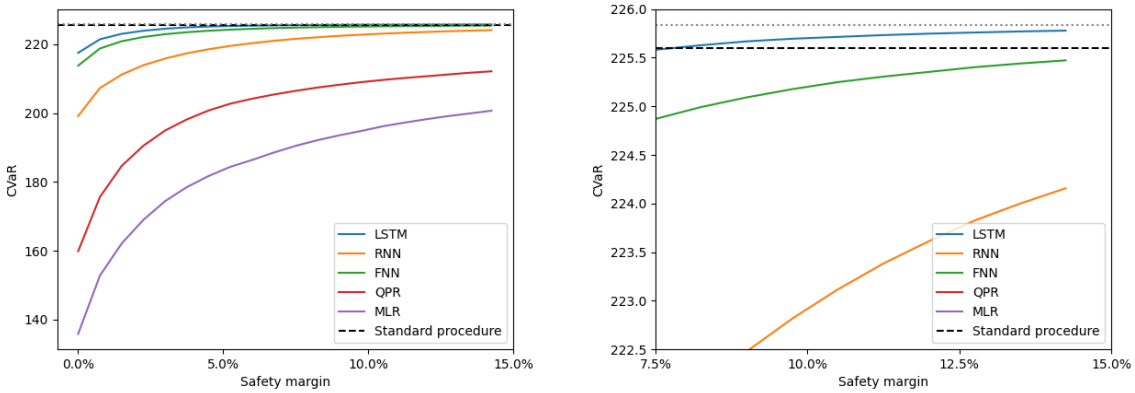
(b) Neural network metamodel

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

Figure 3.5 depicts the average percentage of correctly identified true tail scenarios[5] by different metamodels for different safety margins. The percentage of tail matches of the traditional regression metamodels are significantly lower than the ones of the neural networks. We see that a poor metamodel like the QPR identifies less than 40% of the true tail scenario without any safety margin. In contrast, a good metamodel like the LSTM

[5]Each quantity is averaged over 50 macro replications.

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[6]. The RNN metamodel suffers from the vanishing gradient problem during training. For some macro replications (as shown in Figure 3.2b), the RNN metamodels' predictions are far from the true losses, especially on the tails. 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 reaches similar percentage 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.



(a) Safety margin 0% - 15%.

(b) Safety margin 7.5% - 15%.

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

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

[6]The metamodel in **?** 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

different safety margins, averaged over 50 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 (solid and dotted, respectively) lines in Figure 3.6. We observe that, with reasonable safety 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 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 value 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 Deep Neural Network Metamodels

For financial and actuarial applications, regulators and practitioners are often concerned about the robustness of deep neural network 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 before the injection. In our numerical experiments, we are able to use Monte Carlo simulation 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 crucial question of whether deep learning models are 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 outer scenarios (M) and inner replications (N) used to generate the training data. 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.

Model	**Noise level**	**Training error**	**Test error**	**True error**
LSTM	$N = 100$	0.075($\pm$0.004)	0.079($\pm$0.005)	0.063($\pm$0.004)
High-capacity LSTM	$N = 100$	0.068($\pm$0.004)	0.102($\pm$0.006)	0.060($\pm$0.004)
LSTM	$N = 10$	0.195($\pm$0.001)	0.193($\pm$0.001)	0.070($\pm$0.001)
High-capacity LSTM	$N = 10$	0.157($\pm$0.002)	0.199($\pm$0.002)	0.065($\pm$0.001)
LSTM	$N = 1$	1.366($\pm$0.006)	0.781($\pm$0.005)	0.129($\pm$0.002)
High-capacityLSTM	$N = 1$	1.354($\pm$0.006)	0.795($\pm$0.005)	0.149($\pm$0.002)

Table 3.2: MSEs of LSTM metamodels.

Table 3.2 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. The test errors are included as a practical measure of the metamodels' generalization ability. 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 relationship from the low-noise and medium-noise datasets. However, when the noise level is high, the high-capacity LSTM metamodel has extremely high training error and true error. It also has wide confidence intervals, which is an indication that the high-capacity LSTM metamodel is starting to overfit 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 available. The test error approximates the true error and is directly observable. The test errors of the LSTM metamodels are consistent with their true errors, which suggests that generalization ability of a metamodel can indeed be inferred from its test error.

Another way to read Table 3.2 is to compare the errors of two metamodels for the same noise level. Comparing the LSTM with the high-capacity LSTM on each noise level,

we observe that the high-capacity LSTM has lower training errors and true errors than the LSTM on both the low-noise and medium-noise datasets. However, when the noise level is high, the high-capacity LSTM has lower training errors but higher true errors than the LSTM. In our numerical example, only the highest level of noise moves one towards choosing the LSTM over the high-capacity LSTM. In most cases, a high-capacity metamodel is better. Nevertheless, in extreme circumstances when the noise level is too high, using a high capacity metamodel leads to severe overfitting 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 right metamodel.

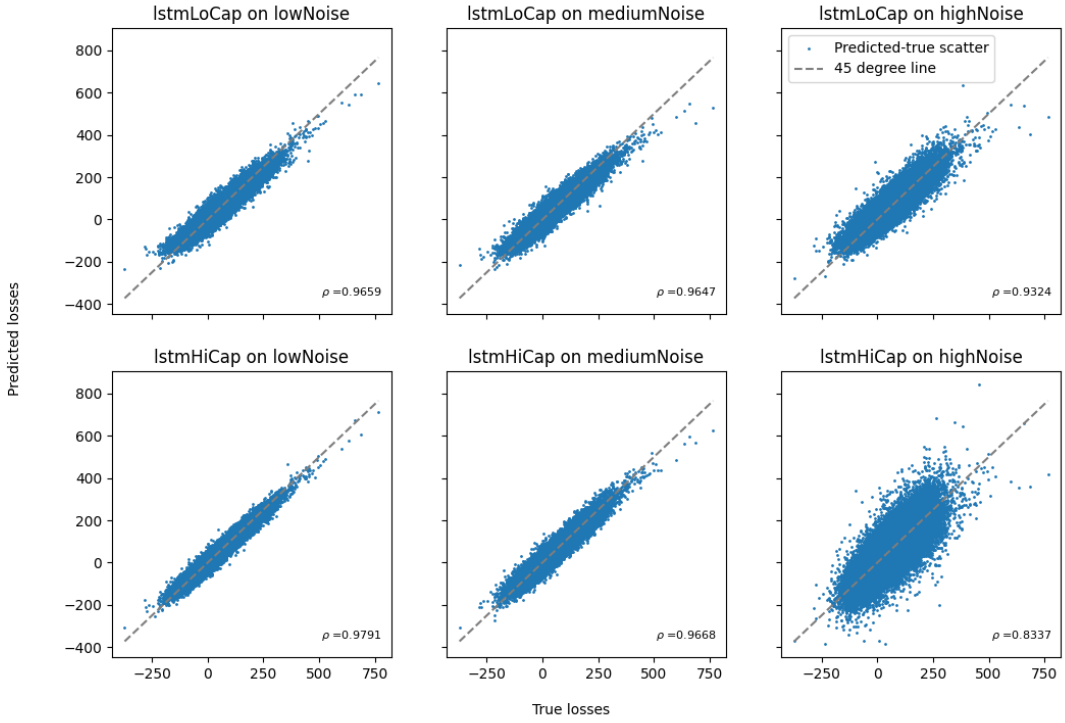


Figure 3.7: QQ-plots between true losses (x-axis) and predicted losses (y-axis) for two LSTM metamodels.

With more 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,

and the three columns correspond to the three 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. Compared to that of the regular LSTM, the metamodel predictions of high-capacity LSTM align more closely with the true losses when trained on the low-noise dataset. 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 picture of the noise tolerance of deep neural network 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 data. The number of outer scenarios corresponds to the number of feature-label pairs (i.e., data size) of the training dataset, and the number of inner replications controls the noise level in the training labels.

	$N = 1{,}000$	$N = 100$	$N = 10$	$N = 1$
$M = 100$	0.158	0.167	0.229	1.139
$M = 1{,}000$	0.127	0.123	0.173	0.559
$M = 10{,}000$	0.097	0.099	0.115	0.283
$M = 100{,}000$	0.063	0.063	0.068	0.126

Table 3.3: MSE between regular LSTM predicted losses and true losses.

Table 3.3 and 3.4 show the average squared errors between the LSTM predictions and the true losses for various combinations of M and N. The last rows of Table 3.3 and Table 3.4 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 significant for a regular LSTM when N is larger than 10. For the high-capacity LSTM, the MSE is significantly reduced when N is increased from 1 to 100, but the reduction is not significant when N is increased

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

Table 3.4: MSE between high-capacity LSTM predicted losses and true losses.

from 100 to $1,000$. Another way to interpret the results in Table 3.3 is to compare the MSEs for the same budget of $\Gamma = M \times N$. Entries on the same diagonal represents the same simulation budget. For most budgets, the MSEs are also the lowest when $N = 10$. The results in Table 3.3 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 **?**, where the authors show that the performance of a regression-based nested simulation procedure is more affected by the number of outer scenarios.

To further investigate the LSTM metamodels' sensitivity to data quantity and quality, we report the Spearman rank correlation coefficients Table 3.5, 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. The p-values of all quantities are less than 10^{-5}, indicating that the correlations are statistically significant.

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

Table 3.5: Spearman (and Pearson) correlation coefficients between regular LSTM predicted losses and true losses.

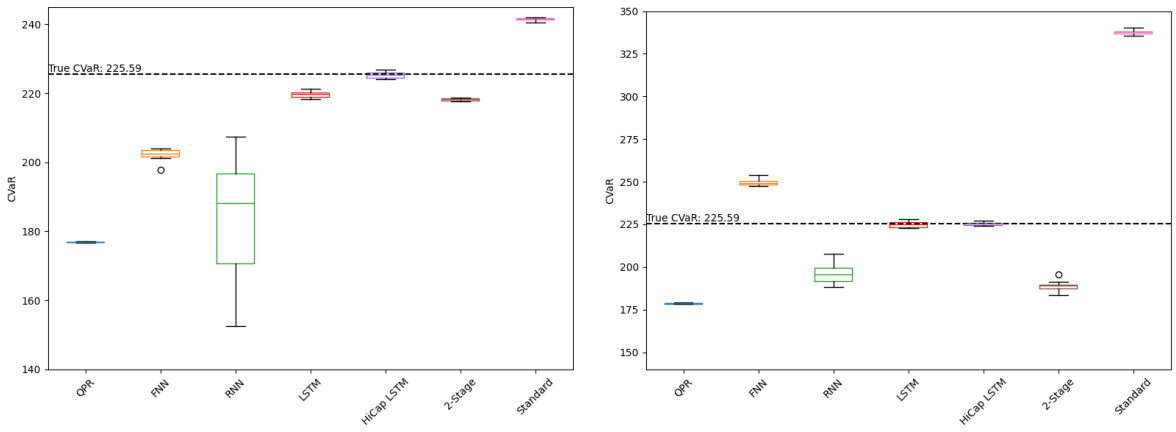
Consistent with the results in Table 3.3, the Pearson and Spearman correlation of the LSTM metamodel predictions with the true losses is sensitive to the number of outer

scenarios, especially at higher noise levels. Notably, the Spearman correlation coefficients are not significantly different from the Pearson correlation coefficients for any N larger than 10. In our numerical experiments, the regular LSTM metamodel is trained on $M = 100,000$ samples from the low noise training dataset ($N = 100$), the total simulation budget is $\Gamma = 10,000,000$. The corresponding Spearman correlation coefficient is 0.927, which is very close to the Pearson correlation coefficient of 0.922. 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 ranking 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 is more efficient and extends to estimating other risk measures that require knowledge of the entire loss distribution. 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 metamodels predictions are used to estimate the risk measures directly.

Figure 3.8 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 extra budget for extensive inner simulation on the predicted tail scenarios in Stage 2. The metamodel with the best performance in the two-stage procedure is chosen, and the safety margin is set to 0%. The standard procedure shown in Figure 3.8 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 LSTM metamodels consistently produce CVaR estimates that are closer to the true value than the standard procedure's estimate, especially when the noise level is high in the training labels. It is another strong evidence that the 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



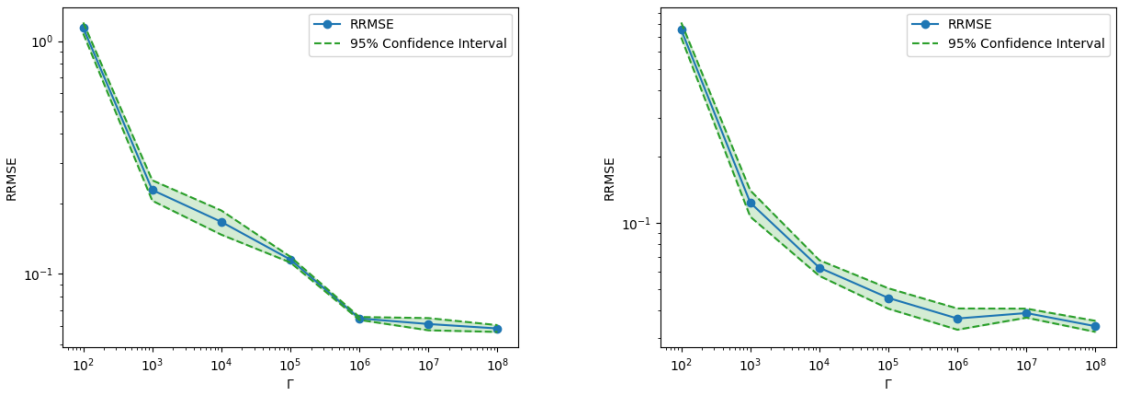
(a) Medium noise training labels ($N = 10$) (b) High noise training labels ($N = 1$)

Figure 3.8: CVaR estimates of the single-stage procedure with metamodels.

among the metamodels is more pronounced in the single-stage procedure than in the two-stage procedure. Trained using low noise and medium noise labels, high-capacity LSTM metamodel consistently produces CVaR estimates that are closer to the true value than the regular LSTM, while the other metamodels produce even worse estimates. 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 3.8 suggest that the FNN metamodel should not be used in the single-stage procedure.

A single-stage procedure with a high-capcity LSTM metamodel is particularly superior to the two-stage procedure, as it is able to achieve a similar accuracy as the standard procedure with a much smaller 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 interval of training time of the high-capacity LSTM metamodel is (19.64 ± 0.94) minutes on a 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.



(a) regular LSTM (b) high-capacity LSTM

Figure 3.9: Empirical convergence of CVaR for the single-stage procedure with LSTM metamodels.

To further investigate the single-stage procedure's performance, we conduct a convergence analysis on the LSTM metamodels. Figure 3.9 shows the log-log plots of the relative root mean square errors (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 suggests 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.

Figure 3.10 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 ($N =$

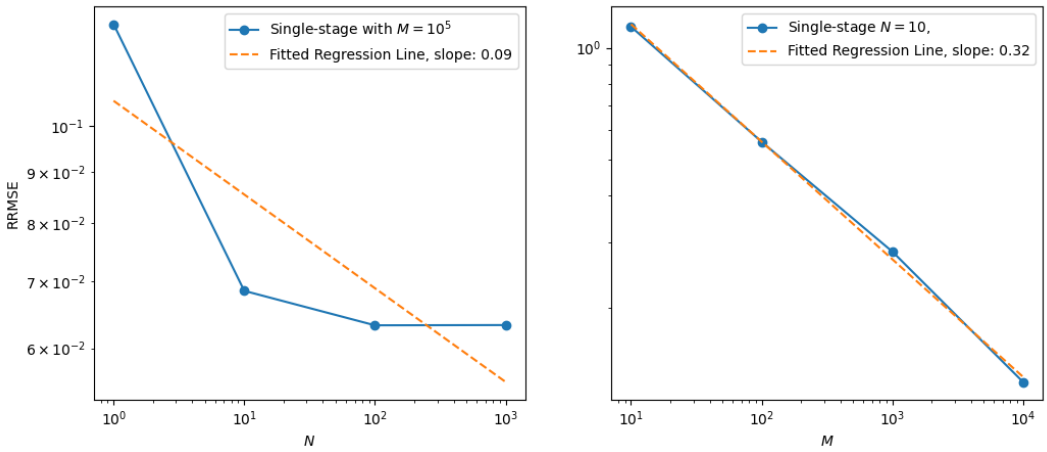


Figure 3.10: Empirical convergence of the single-stage procedure with a LSTM metamodel.

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 $N = 1,000$. 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 3.10. For an increasing number of inner replications with fixed number of outer scenarios, the RRMSE cease to decrease after reaching $N = 100$. More specfically, when the quality of the training labels is fixed at $N = 10$, the CVaR estimator of single-stage procedure with a LSTM metamodel converge roughly in the order of $\mathcal{O}(M^{\frac{2}{3}})$. Hence, in practice, we suggest to run 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 deep neural network metamodels 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 deep neural network models. We find that a deep neural network 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 to influence regulatory bodies towards recognizing 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 a promising evidence that deep neural network 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 deep neural network 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 use them to identify the potential tail scenarios. An extensive inner simulation is performed 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 deep neural network metamodels result in further computational savings over our two-stage procedure. Furthermore, our numerical results provide evidences for adopting deep neural network metamodels in Monte Carlo simulation for risk management tasks. Through our systematic study of the noise tolerance of deep neural network metamodels, we address regulatory concerns by showing that a LSTM metamodel 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 deep neural network 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 deep neural network 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 opti-

mal network architecture may vary for different simulation models. Exploring deep neural network 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.

Chapter 4

Efficient Transfer of Knowledge for Deep Hedging of Variable Annuities

4.1 Introduction

Reinforcement Learning (RL) is a branch of machine learning that focuses on training algorithms, known as agents, to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment by trial and error, using feedback from its own actions and experiences. Unlike supervised learning, where training data is labeled with the correct answers, in RL, an agent is provided with rewards or punishments as signals for its actions.

The core of RL revolves around the concept of the agent interacting with its environment over time, aiming to maximize the cumulative reward. This process involves observing the state of the environment, selecting and performing actions, and receiving rewards or penalties in response to those actions. The agent's objective is to learn a mapping from states to actions that maximizes this cumulative reward, which is often referred to as a policy. One of the fundamental frameworks for modeling RL problems is the Markov Decision Process (MDP). An MDP provides a mathematical formulation of the decision-making process, characterized by states, actions, rewards, and transition probabilities. Solving an MDP involves finding a policy that maximizes some function of the expected rewards, typically the expected cumulative reward over time.

Reinforcement learning algorithms can be broadly categorized into two types: model-based and model-free approaches. Model-based methods utilize a model of the environment to simulate the outcomes of actions, enabling planning and decision-making with

fewer interactions with the environment. Conversely, model-free methods learn directly from interactions with the environment, without relying on a model, making them more straightforward but often less efficient in terms of sample usage.

A compelling application of RL is in the hedging of financial derivatives, a domain where the complexity and uncertainty of financial markets make traditional static models inadequate. Financial derivatives are contracts whose value is derived from an underlying asset, and hedging is the practice of making investments to reduce the risk of adverse price movements. RL's adaptability and learning capabilities offer a promising solution to dynamically adjust hedging strategies in response to market movements. This approach, particularly with model-free algorithms, has the potential to enhance risk management practices by developing strategies that can adapt in real-time to changing market conditions.

In this paper, we propose a transfer learning approach to improve the risk management of variable annuities (VAs) by learning a hedging strategy utilizing previous experiences from other financial products. VAs are popular retirement products that combine investment and insurance features, offering policyholders the potential for investment growth and the protection of a guaranteed minimum death benefit or income benefit. Due to the complexity of the products and the dynamic nature of the financial markets, the task of hedging VAs is particularly challenging. We aim to leverage the knowledge from the hedging strategies of financial derivatives to construct hedging strategies for VAs, thereby improving the risk management of these products. The rest of the paper is organized as follows: Section 4.2 presents the problem formulation for hedging variable annuity with a deep neural network, Section 4.3 presents the background knowledge for transfer learning, and Section 4.4 states the plans for ongoing numerical experiments.

4.2 Dynamic Hedging of Variable Annuities

4.2.1 Variable Annuities

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

Two relevant types of guarantees in our studies are:

- **Guaranteed Minimum Maturity Benefit (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.

- **Guaranteed Minimum Withdrawal Benefit (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 GMDB is usually offered. Additional features offered with the GMWB include roll-up, ratchet, and reset (**?**).

For a comprehensive review of other types of VA contracts such as Guaranteed Minimum Death Benefit (GMDB), Guaranteed Minimum Accumulation Benefit (GMAB) and Guaranteed Lifetime Withdrawal Benefit (GLWB), we refer readers to **?**. Next we present a summary of dynamic hedging for VA contracts. We refer readers to **?** 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 $\tau > 0$. 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. Note that the policyholder's (random) time of death also affects the timing of the benefit payout for certain types of VA such as GMDB, 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 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 sub-account value changes according to the growth of the underlying stock and

the (gross) management fee is deducted. That is,

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

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., $F_t \eta_n$.

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

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

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

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

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) whose evolution is driven by the stochasticity of S_t. In practice, the simulation model may also incorporate additional complications like 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 GMMB contract is $L_t = e^{-r(T-t)} \cdot (G_T - F_T)^+ - \sum_{s=t+1}^{T} e^{-r(T-s)} F_s \eta_n$. Also, the insurer's time-t liability for a GMWB contract is $L_t = \sum_{s=t+1}^{T} e^{-r(T-s)} [(I_s - F_s)^+ - \eta_n F_s]$.

For example, consider the time-t liability L_t of a GMWB: Suppose that given the stock sample path, e.g., an outer path $S_1, \ldots, S_t$, one can simulate future stock prices $\widetilde{S}_{t+1}, \ldots, \widetilde{S}_T$, e.g., inner sample paths, based on some asset model such as a Black-Scholes model. The tilde symbol ($\sim$) over a quantity denotes its association with the inner simulation. Given the time t state (S_t, F_t, G_t), following **?** the sensitivity of L_t with respect to S_t can be estimated by a pathwise estimator (**?**):

$$\Delta_t(\widetilde{S}_{t+1}, \ldots, \widetilde{S}_T | S_t) = \frac{\partial L_t}{\partial S_t} = \sum_{s=t+1}^{T} e^{-r(T-s)} \left[\mathbf{1}\{\widetilde{I}_s > \widetilde{F}_s\} \cdot \left(\frac{\partial \widetilde{I}_s}{\partial S_t} - \frac{\partial \widetilde{F}_s}{\partial S_t} \right) - \eta_n \frac{\partial \widetilde{F}_s}{\partial S_t} \right], \quad t = 0, \ldots, T-1, \tag{4.1}$$

where $\mathbf{1}\{\cdot\}$ is an indicator function and

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

The recursion is initialized with $(\widetilde{S}_t, \widetilde{F}_t, \widetilde{G}_t) = (S_t, F_t, G_t)$, $\frac{\partial \widetilde{F}_s}{\partial S_t} = \frac{\widetilde{F}_t}{S_t}$, and $\frac{\partial \widetilde{G}_s}{\partial S_t} = \frac{\partial \widetilde{I}_s}{\partial S_t} = 0$.

4.2.2 Markov Decision Process (MDP) for Hedging VAs

A Markov Decision Process (MDP) provides a mathematical framework for solving sequential decision-making tasks in situations where outcomes are partly random and partly under the control of an agent. In the context of hedging VAs, an MDP can be used to model the decision-making process of an insurer who aims to minimize the liability of the VA by dynamically adjusting the hedging portfolio based on the current state of the VA and the financial markets. An MDP is defined by the tuple $\mathcal{M} = (\mu_0, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- μ_0 is the initial states.
- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is a state transition probability distribution, where $\mathcal{P}(s'|s, a)$ represents the probability of transitioning from state s to state s' due to action a.
- $\mathcal{R}$ 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, $\pi : \mathcal{S} \to \mathcal{A}$, that maximizes the cumulative reward over time, often referred to as the value function for a policy π at state s:

$$V_{\mathcal{M}}^{\pi}(s) = \mathbb{E}_{s_0 \sim \mu_0, a_i \sim \pi(\cdot|s_i), s_{i+1} \sim \mathcal{P}(\cdot|s,a)} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | \pi, s \right], \tag{4.2}$$

where $R_{t+1} = \mathcal{R}(s_i, a_i, s_{i+1})$ is the reward received by the agent after taking action a_i in state s_i and transitioning to state s_{i+1}, and the expectation is taken over the possible sequences of states and rewards that follow from the policy π. In the context of hedging VAs, the initial states μ_0 specifies the initial value of the VA and the market specifications. The state space $\mathcal{S}$ represent the information available to the agent, such as current value of the underlying asset (S_t), the current subaccount value (F_t), and the guarantee value (G_t), as well as other relevant contract specifications. The action space $\mathcal{A}$ represent the hedging decisions, such as the hedging portfolio weights. The state transition probability distribution $\mathcal{P}$ and the reward function $\mathcal{R}$ are derived from the dynamics of the VA and the market specifications. Instead of having an infinite time horizon, the final reward is obtained at the maturity of the VA contract. Hence, $R_t = 0$ for all $t > T$. The discount factor γ models the present value of future rewards and is conveniently inherited from the risk-free rate.

Considering the task of hedging a GMWB contract, the hedging strategy aims to minimize the insurer's liability L_t at each time $t = 1, \ldots, T$ by dynamically adjusting the hedging portfolio based on the current state of the VA and the financial markets. The objective is to find a policy π that minimizes the expected liability over time, i.e.,

$$\min_{\pi} \mathbb{E}[\sum_{t=1}^{T} \gamma^t L_t | \pi, s_0], \tag{4.3}$$

where it can be conveniently rewritten as the value function $V_{\mathcal{M}}^{\pi}(s)$ in Equation 4.2 by setting the reward function $R_t = -L_t$ and the discount factor $\gamma = \frac{1}{1+r}$. Other value functions can be used to optimize over a risk measure on the cumulative liability, such as the Value-at-Risk (VaR) or the Conditional Value-at-Risk (CVaR). Denote the risk measure as ρ_α, where α is the confidence level, then the value function can be set as

$$V_{\mathcal{M}}^{\pi}(s) = \rho_\alpha(\sum_{t=1}^{T} \gamma^t L_t | \pi, s_0). \tag{4.4}$$

4.2.3 Deep Hedging Approaches

Deep hedging is a reinforcement learning approach that leverages deep learning models to optimize hedging strategies for financial derivatives. In deep hedging, the objective is to minimize the hedging error or a risk measure over a set of paths of the underlying asset, which involves training the DNNs using backpropagation and optimization algorithms to find the hedging strategy that maximizing the value function.

1. **Value Function:** Specify a value function that quantifies the hedging performance. This could include minimizing the expected liability, the CVaR, or the variance of portfolio returns.

2. **Training and Optimization:** Using historical data or simulated data generated from the specified market model, train the DNN using backpropagation and optimization algorithms to find the hedging strategy that maximizes the value function.

3. **Implementation and Adjustment:** Implement the learned hedging strategy in real-time trading, with periodic adjustments based on new market information and continuous learning to adapt to changing market dynamics. This is often referred to as online learning.

Depending on the specific market model and the hedging objective, deep hedging can be categorized into model-based and model-free approaches. Model-based deep hedging trains a deep neural network (DNN) to learn optimal hedging actions by maximizing a value function that reflects the hedging error or risk measure, such as the variance of the hedging portfolio's final value or tail risk measures like the CVaR. The DNN processes sequential market data and outputs a policy that determines the hedging actions based on the current state of the market. In the context of deep hedging, the policy $\pi = \phi(\mathcal{I})$ is defined by the parameter ϕ of a DNN, and the information $\mathcal{I}$ is the training data during the learning process. Buehler et al. (2019) is the first paper to propose a model-based deep hedging approach for hedging financial derivatives with a focus on minimizing CVaR. A more recent paper by Imaki et al. (2021) proposes a new network architecture for model-based deep hedging that can be trained more efficiently. A typical setup for a model-based deep hedging strategy includes:

4a. **Market Model Specification:** Define the underlying asset price dynamics and the risk factors affecting the financial instruments. Common models include geometric Brownian motion for stock prices or more complex models accounting for jumps or

stochastic volatility. **?** provides detailed discussions on simulating asset prices under various models.

5a. **Deep Neural Network Design:** Design a DNN architecture capable of processing sequential market data and making hedging decisions. The network outputs a policy by receiving inputs that include historical prices, option strikes, and other relevant financial indicators. Buehler et al. (2019) uses a feedforward neural network (FNN), where its input layer receives the current state of the market, and its output layer produces the hedging weights.

Conversely, model-free deep hedging approaches directly learn from historical market data without making explicit assumptions about asset price dynamics. Instead, the value functions are learned directly from the data, and the hedging strategies are derived from the learned value functions. A typical setup for a model-free deep hedging strategy includes:

4b. **Market Model Specification:** No explicit model assumptions are made about the underlying asset price dynamics. Instead, the deep learning model learns the value function and hedging strategies directly from historical market or simulation data.

5b. **Deep Neural Network Design:** Design two DNNs: a value network and a policy network. The value network learns the value function, while the policy network learns the optimal hedging actions based on the approximated value function.

A model-free approach does not require explicit model assumptions about asset price dynamics, making it more flexible and adaptable to different market conditions and more robust to model misspecification.

4.2.4 Value-based and Policy-based Deep Reinforcement Learning

Model-free deep reinforcement learning algorithms can be broadly categorized into two types: value-based and policy-based approaches.

1. **Value-based Deep Reinforcement Learning:** 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 Black-Scholes framework.

2. **Policy-based Deep Reinforcement Learning:** 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. Representative algorithms include the Proximal Policy Optimization (PPO) algorithm proposed by Schulman et al. (2017) and the Trust Region Policy Optimization (TRPO) algorithm proposed by Schulman et al. (2015). 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.

4.3 Transfer Learning for Risk Management of Variable Annuities

Transfer learning is a machine learning technique that leverages knowledge from one domain to improve learning in another domain. In the context of deep reinforcement learning, transfer learning can be used to improve the risk management of VAs in five ways:

1. **Transferring to VA products**: Knowledge can be transferred from the hedging strategies of financial derivatives to constructing hedging strategies for VAs. VAs are complex insurance products with unique features, which makes it challenging to develop effective hedging strategies. However, the payoff structures of VAs are similar to those of financial derivatives, such as European options, which have been extensively studied in the literature. By leveraging the knowledge from the hedging strategies of financial derivatives, we can improve the risk management of VAs.

2. **Transferring to new VA products**: Knowledge can be transferred from the hedging strategies of one VA product to another VA product. Different VA products have different features and payoff structures, which makes it challenging to develop hedging strategies for new VA products. However, experience from hedging one VA product can be transferred to develop hedging strategies for new VA products, especially when the new VA product is similar to the existing ones.

3. **Transferring to new asset models**: Knowledge can be transferred from the hedging strategies of one asset model to another asset model. A stochastic volatility model is a more realistic model for asset prices than the Black-Scholes model, but it is more challenging to develop hedging strategies for VAs under such a model.

4. **Transferring to new model specifications**: Model misspecification is a common issue in financial modeling, and training on data simulated with misspecified parameters can lead to suboptimal hedging strategies. If only the misspecified model is similar to the correct model, then we can leverage the knowledge from the misspecified model to improve the hedging strategies under the correct model, since the only difference between the two models is the parameter values.

5. **Transferring to the real world**: Knowledge can be transferred from simulated data to real-world data. In practice, it is challenging to obtain real-world data for VAs, but we can leverage simulated data to develop hedging strategies for VAs. If the simulated data is representative of the real-world data, then the hedging strategies developed from the simulated data can be transferred to the real world.

Given a set of source domains $\boldsymbol{\mathcal{M}}_s = \{\mathcal{M}_s | \mathcal{M}_s \in \boldsymbol{\mathcal{M}}_s\}$ and a target domain $\mathcal{M}_t$, transfer learning aims to improve the learning on the target domain. While utilizing the interior information $\mathcal{I}_t$ from the target domain, the exterior information $\mathcal{I}_s$ from the source domains is also leveraged to improve the learning on the target domain.

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s\sim\mu_0, a\sim\pi} \left[V^{\pi}_{\mathcal{M}}(s,a) \right], \tag{4.5}$$

where $\pi = \phi(\mathcal{I}_s \sim \boldsymbol{\mathcal{M}}_s, \mathcal{I}_t \sim \mathcal{M}_t)$ is policy learned from target domain based on information from both the source domains and the target domain. A regular reinforcement without transfer learning is a special case where $\mathcal{I}_s = \emptyset$ and $\pi = \phi(\mathcal{I}_t \sim \mathcal{M}_t)$.

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.

4.3.1 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 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).

4.3.2 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

$$\mathcal{J}(\pi_\phi, \pi_{E_i}) = \sum_{t=0}^{\infty} \mathbb{E}_{s_t \sim \mu_0^t, a_t \sim \pi_\phi} \left[\gamma^t R_{t+1} + \frac{\alpha}{\beta} \log \pi_\phi(a_t|s_t) - \frac{1}{\beta} log \pi_{E_i}(a_t|s_t) \right], \tag{4.6}$$

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.

4.3.3 Representation Transfer

Representation transfer is a technique that transfers the knowledge based on the following critical assumption:

Assumption 14 *The state space $\mathcal{S}$, the action space $\mathcal{A}$, and the reward function $\mathcal{R}$ can be disentangled into orthogonal subspaces, which are task-invariant such that knowledge can be transferred between domains on the universal subspaces.*

A suitable representation transfer technique for our task of hedging VAs is modular networks (Andreas et al., 2017), which decomposes a policy π into two sub-modules g and f, i.e.,

$$\pi = \phi(s_{market}, s_{guarantee}) = f(g(s_{market}), s_{guarantee}), \tag{4.7}$$

Zhang et al. (2018) uses modular networks to transfer knowledge of physics models from previous tasks to new tasks in a simulated environment. In the context of hedging VAs, g is the module that processes the market information, and f is the module that processes the guarantee information. The core idea behind modular networks is that the market information can be applied for hedging strategies across different VA products, while the guarantee information is specific to each VA product. Hence, knowledge can be transferred from the source domains to the target domain by transferring the market module g.

4.3.4 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:

- **Asymptotic performance:** the final reward of the agent in the target domain.
- **Transfer ratio:** the ratio of the agent's performance in the target domain with and without transfer learning.

Generalization, on the other hand, refers to the agent's ability to quickly adapt its learned knowledge to the target domain.

- **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.

4.4 Plans for Ongoing Numerical Experiments

As we aim to enhance the risk management strategies for variable annuities (VAs) through the application of transfer learning, our next step involves conducting a series of numerical experiments. These experiments are crucial for validating the theoretical frameworks and methodologies developed so far. Our plan includes:

1. **Simulation setup:** Designing a simulation environment that closely mirrors real-world market dynamics and VA contract specifics. This setup will include various asset models, including the Black-Scholes and stochastic volatility models, to test the adaptability of transferred hedging strategies across different market conditions.

2. **Baseline model comparison:** Establishing a set of baseline hedging strategies against which we can compare the performance of our transfer learning-enhanced strategies. This will likely involve traditional delta hedging techniques and possibly other reinforcement learning approches that have not utilized transfer learning.

3. **Transfer learning implementation:** Applying the transfer learning methodologies discussed, including reward shaping, policy transfer, and/or representation transfer to the task of hedging VAs. The transfer can happen between different VA products, different asset models, different model specifications, and between simulated and real-world data.

4. **Performance evaluation:** Utilizing the evaluation metrics outlined in the previous sections to assess mastery and generalization of the transfer learning strategies. Key metrics will include asymptotic performance, transfer ratio, jumpstart performance, and time to threshold.

5. **Sensitivity analysis:** Conducting sensitivity analyses to understand how changes in market conditions, contract specifics, and model parameters affect the performance of our transfer learning strategies. This will help in identifying the robustness and limitations of our approach.

6. **Hedging in real-world:** If accessible, applying the trained models to real-world data to further validate their practical applicability and performance in real-world market conditions.

Through these numerical experiments, we aim to demonstrate the efficacy of transfer learning in improving the hedging of VAs in a variety of market conditions and contract

specifics. We expect that our transfer learning-enhanced strategies will outperform traditional hedging techniques and possibly other machine learning approaches that have not utilized transfer learning. This will provide strong evidence for the potential of transfer learning in improving risk management strategies for VAs and other complex financial products in a dynamic market environment.

4.4.1 Initial Numerical Experiments

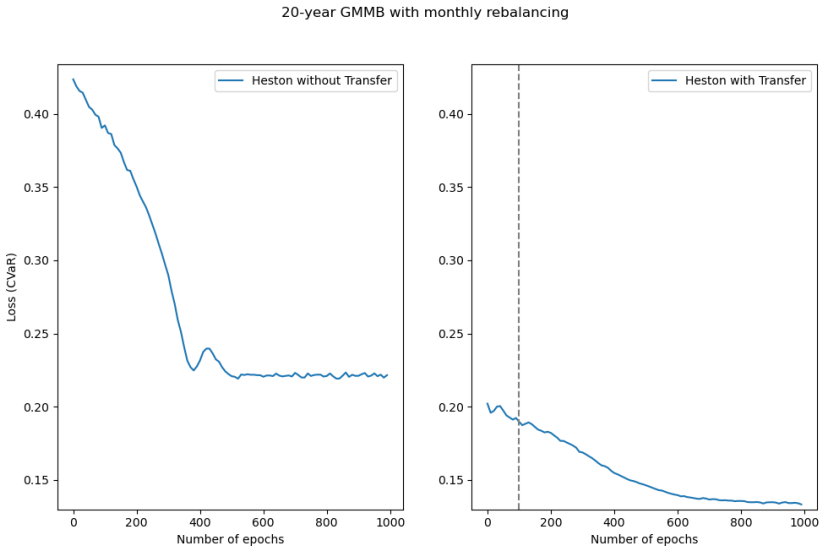


Figure 4.1: Initial numerical experiments for GMMB

Bibliography

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International conference on machine learning*, pages 166–175. PMLR, 2017.

Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, 19 (8):1271–1291, 2019.

Wing Fung Chong, Haoen Cui, and Yuxuan Li. Pseudo-model-free hedging for variable annuities via deep reinforcement learning. *Annals of Actuarial Science*, 17(3):503–546, 2023.

Anna Harutyunyan, Sam Devlin, Peter Vrancx, and Ann Nowé. Expressing arbitrary reward functions as potential-based advice. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

Shota Imaki, Kentaro Imajo, Katsuya Ito, Kentaro Minami, and Kei Nakagawa. No-transaction band network: A neural network architecture for efficient deep hedging. *arXiv preprint arXiv:2103.01775*, 2021.

Petter N Kolm and Gordon Ritter. Dynamic replication and hedging: A reinforcement learning approach. *The Journal of Financial Data Science*, 1(1):159–171, 2019.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321, 1992.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer, 1999.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Eric Wiewiora, Garrison W Cottrell, and Charles Elkan. Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 792–799, 2003.

Bo Xiao, Wuguannan Yao, and Xiang Zhou. Optimal option hedging with policy gradient. In *2021 International Conference on Data Mining Workshops (ICDMW)*, pages 1112–1119. IEEE, 2021.

Wei Xu and Bing Dai. Delta-gamma-like hedging with transaction cost under reinforcement learning technique. *The Journal of Derivatives*, 29(5):60–82, 2022.

Amy Zhang, Harsh Satija, and Joelle Pineau. Decoupling dynamics and reward for transfer learning. *arXiv preprint arXiv:1804.10689*, 2018.