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

Tony Wirjanto Mingbin Feng Xintong Li

Abstract

Deep neural network models have gained great success in many applications, but their adoption in financial and actuarial applications has been received by regulators with some trepidation. The lack of transparency and interpretability of these models leads to skepticism about their resilience and reliability, which are important factors to ensure financial stability and insurance benefit fulfillment. In this study, we use stochastic simulation as a data generator to examine deep neural networks under controlled settings. Our study shows interesting findings in fundamental questions like "What do deep neural networks learn from noisy data?" and "How well do they learn from noisy data?", and the findings provides justifications for using deep neural networks in actuarial applications. Based on our findings, we propose an efficient nested simulation procedure that uses deep neural networks as metamodels to estimate risk measures of hedging errors for variable annuities. The proposed procedures use deep neural networks as metamodels to provide accurate loss predictions and concentrate simulation budget on tail scenarios while maintaining transparency in the estimation step, and we provide practical guidelines to extend our generic procedures for other financial and actuarial applications.

1 Introduction

Deep neural networks (Hastie et al., 2009; LeCun et al., 2015) have attracted attentions of researchers and practitioners due to their success in solving real-world supervised learning tasks such as AlphaGo (Silver et al., 2016) and ChatGPT (OpenAI, 2023). Since the first artificial neural network model (McCulloch and Pitts, 1943) and the first algorithm for training a perceptron (Rosenblatt, 1958), especially after the introduction of backpropagation (Rumelhart et al., 1985) and the growth of high-performance computing, the field of artificial neural network and deep learning in general has grown rapidly. Two specialized neural network architectures that are relevant to our study are recurrent neural networks (RNNs) (Williams and Zipser, 1989; Sutskever et al., 2014) and long-short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997; Chung et al., 2014), 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. Poole et al. (2014) show that injecting synthetic noise before and after hidden unit activations during training improves the performance of autoencoders. Neelakantan et al. (2015) improve learning for 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, Luo et al. (2016) show that adding synthetic label noise to the convolutional neural network (CNN) can improve its ability to capture global features. Srivastava et al. (2014) quantify the error tolerance by injecting synthetic label noise with a custom Boltzmann machine hardware. Szegedy et al. (2013) find that neural networks are vulnerable to adversarial examples, and Goodfellow et al. (2014) design an efficient method to generate such noisy examples to exploit the vulnerability to adversarial perturbations. Carlini and Wagner (2017) design targeted attacks to training labels to test the robustness of neural networks. Instead of using synthetic noise, Jiang et al. (2020) inject real-world label noise and examine noise tolerance of neural networks with controlled noise levels. The aforementioned studies use real-world data, as is typically the case for many neural network studies, where noise is already present in the training labels before any noise injection. Users of real-world data have little control over the noise level of the original training labels

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

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. Fonseca et al. (2003) provide general guidelines for simulation metamodeling with neural networks, Lieu et al. (2022) use deep neural networks as metamodels of a simulation model for structural reliability analysis, and Salle and Yıldızoğlu (2014) 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. Liu and Staum (2010) 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 Gan and Lin (2015) use stochastic kriging for an efficient valuation of large portfolios of variable annuity (VA) contracts. Other studies, such as Broadie et al. (2015), Hong et al. (2017), and Zhang et al. (2022) 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: Section 2 presents the problem settings for tail risk measures and dynamic hedging of VAs. Section 3 proposes an efficient two-stage nested simulation procedure that uses deep neural networks as metamodels to help reduce simulation budget by only performing computations on identified tail scenarios. Section 4 proposes a one-stage nested simulation procedure that estimates risk measures directly with metamodel predictions. Section 5 demonstrates the efficiency of deep learning proxies and examines error tolerance of two LSTM metamodels with different numbers of trainable parameters. Practical suggestions are provided for the choice of suitable metamodels and simulation settings.

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 and Section 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.

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) (Rockafellar and Uryasev, 2002). 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: $\text{VaR}_\alpha = q_\alpha = \inf\{q : \Pr(L \le q) \ge \alpha\}$. 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 EIOPA (2014). In Canada, the regulatory capital requirement for VAs is set based on CVaRs as prescribed in OSFI (2017).

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

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 (The Geneva Association, 2013).

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 Hardy (2003). Next we present a summary of dynamic hedging for VA contracts. We refer readers to Dang (2021) for detailed modeling of insurer liabilities in different VA contracts and Greek estimation.

Consider a generic VA contract with maturity $T > 0$ periods, e.g., $T = 240$ months. Denote the policyholder's (random) time of death by $\tau > 0$. 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 Cathcart et al. (2015) the sensitivity of V_t with respect to S_t can be estimated by a pathwise estimator (Glasserman, 2004):

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

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 $\frac{\partial \widetilde{G}_s}{\partial S_t} = \frac{\partial \widetilde{I}_s}{\partial S_t} = 0$.

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.

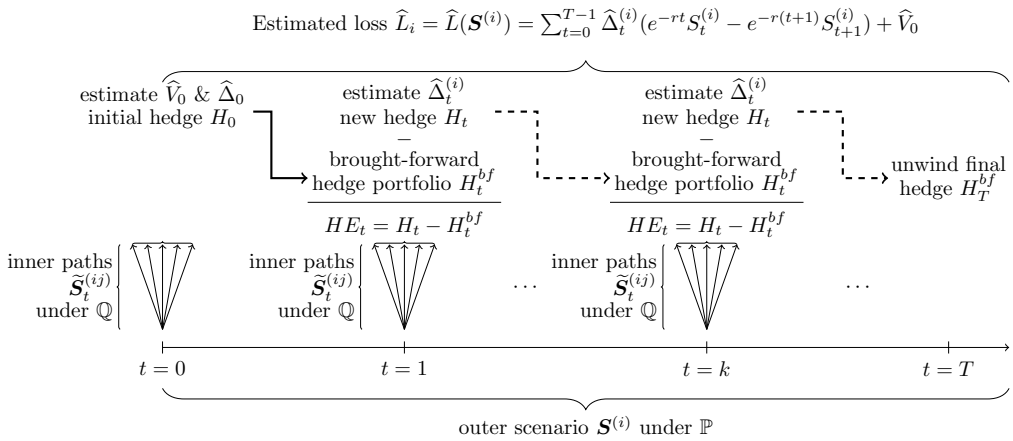


Figure 1: Illustration of multi-period nested simulation that estimates the P&L for one outer scenario.

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 and Section 4 can be trivially adapted to more general hedging strategies.

Consider a generic 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

$$HE_t = H_t - H_t^{bf}, \quad t = 1, \ldots, T-1. \tag{2}$$

The P&L of the VA contract includes the cost of the initial hedge (H_0), the hedging errors (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

$$L = H_0 + \sum_{t=1}^{T-1} e^{-rt} HE_t - e^{-rT} H_T^{bf} + V_0 = \sum_{t=0}^{T-1} \Delta_t (e^{-rt} S_t - e^{-r(t+1)} S_{t+1}) + V_0, \tag{3}$$

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

In (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 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 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.

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 $\widehat{L}_{(1)}^{MC} \le \widehat{L}_{(2)}^{MC} \le \cdots \le \widehat{L}_{(M)}^{MC}$.
4: Estimate α-CVaR of L by $\widehat{\text{CVaR}}_\alpha^{MC} = \frac{1}{(1-\alpha)M} \sum_{i=\alpha M+1}^{M} \widehat{L}_{(i)}^{MC} = \frac{1}{(1-\alpha)M} \sum_{i \in \widehat{\mathfrak{T}}_{(1-\alpha)M}^{MC}} \widehat{L}_i^{MC}$.

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 1, as one inner simulation experiment. We make four observations:

- each inner simulation is time-consuming, as it includes T simulation experiments, one at each time $t = 0, \ldots, T-1$,
- 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 $\widehat{\mathfrak{T}}_k$ are used in the estimator.

3 Two-Stage Nested Simulation with Metamodels

Based on the three observations above and inspired by Dang et al. (2020), 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 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, $\widehat{\mathfrak{T}}_m^{PD}$, 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 $\widehat{\mathfrak{T}}_m^{PD}$. 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 $\widehat{\text{CVaR}}_\alpha^{ML} = \frac{1}{(1-\alpha)M} \sum_{i=\alpha M+1}^{M} \widehat{L}_{(i)}^{ML} = \frac{1}{(1-\alpha)M} \sum_{i \in \widehat{\mathfrak{T}}_{(1-\alpha)M}^{ML}} \widehat{L}_i^{ML}$, where $\widehat{\mathfrak{T}}_k^{ML}$ denotes a predicted tail scenario set associated with the largest k estimated losses.

Similar to Dang et al. (2020), the proposed two-stage procedure in Algorithm 2 uses the metamodel predictions to identify the predicted tail scenario set in Stage 1. However, different from their fixed-budget simulation design, we attempt to achieve a target accuracy. Specifically, in Stage 2 we propose using a standard procedure with the same number of inner replications, N, as a 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 $\widehat{\mathfrak{T}}_m^{PD}$ and are ultimately included in $\widehat{\mathfrak{T}}_{(1-\alpha)M}$ 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.

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

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.

5 Numerical Results

We conduct a series of simulation experiments to (1) demonstrate the efficiency of the proposed metamodeling procedures and (2) examine the error tolerance to noisy training data in deep learning models. The problem settings in our experiments are identical to those in Dang et al. (2020): we consider estimating the 95% CVaR of the hedging loss of a GMWB contract, which is one of the most complex VA contracts in the market. The VA contracts have a 20-year maturity and are delta-hedged with monthly rebalancing, i.e., $T = 240$ rebalancing periods. The gross and net management fees are $\eta_g = 0.2\%$ and $\eta_n = 0.1\%$, respectively. The withdrawal rate for GMWB is 0.375% per month. 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 Dang et al. (2020).

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 $\widehat{\mathrm{CVaR}}_{95\%}$ and denote the corresponding *true tail*

scenario set by Υ_{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 (Kingma and Ba, 2014) 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 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**	**# parameters**	**Training error**	**Test error**	**True error**
MLR	N/A	241	0.707($\pm$0.001)	0.691($\pm$0.003)	0.694($\pm$0.001)
QPR	N/A	481	0.544($\pm$0.001)	0.544($\pm$0.003)	0.530($\pm$0.001)
FNN	$240, 128, 16$	35,009	0.118($\pm$0.010)	0.291($\pm$0.006)	0.113($\pm$0.009)
RNN	$(240, 32), (240, 4), 32$	32,021	0.152($\pm$0.063)	0.167($\pm$0.067)	0.150($\pm$0.063)
LSTM	$(240, 32), (240, 4), 32$	35,729	0.082($\pm$0.006)	0.093($\pm$0.006)	0.070($\pm$0.006)

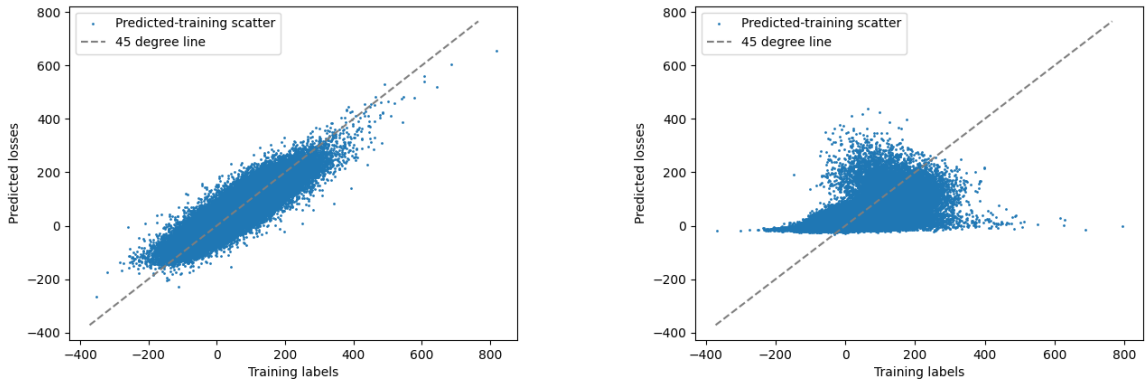
Table 1: Architectures and MSEs of metamodels for GMWB inner simulation model.

0.03225 For each metamodel, 50 independent macro replications are run[2]. The last three columns in Table 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. 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 2 shows the quantile-quantile (QQ) plots between the **training** labels and the RNN predictions for two macro replications, where training of the latter is hindered by the vanishing gradient problem. As a result, the half length of the 95% confidence interval of the RNN's training error is much larger than all other metamodels.

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

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.

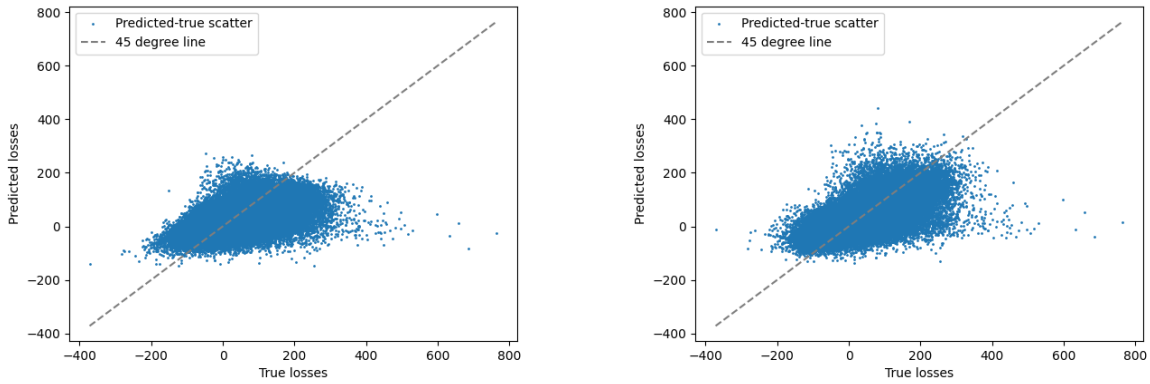


(a) A good RNN metamodel

(b) A bad RNN metamodel

Figure 2: QQ-plots between training labels (x-axis) and predicted losses (y-axis) for the RNN metamodel.

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 4 and Figure 4 shows the QQ plots between the **true** labels and the predicted losses for the regression metamodels and the neural network metamodels, respectively.



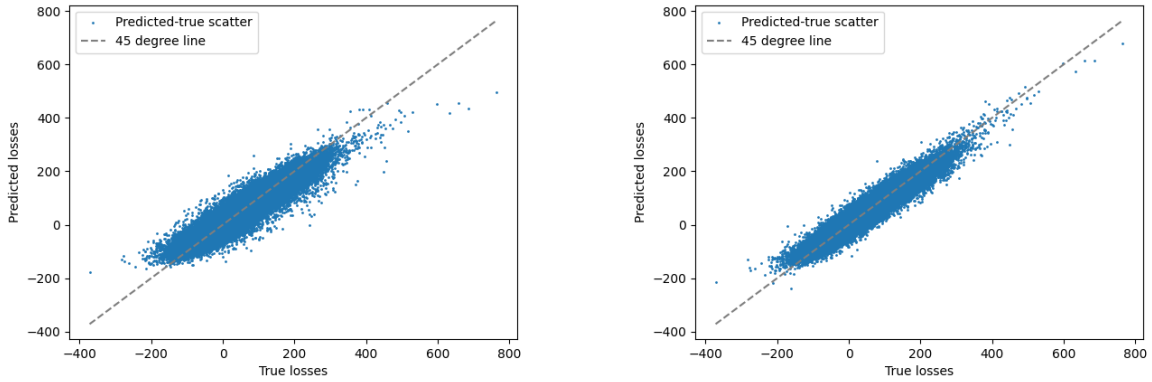
(a) MLR metamodel

(b) QPR metamodel

Figure 3: QQ-plots between true losses (x-axis) and predicted losses (y-axis) for regression metamodels.

Compared with the MSE table in Table 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 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 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 4: QQ-plots between true losses (x-axis) and predicted losses (y-axis) for neural network metamodels.

In Figure 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.

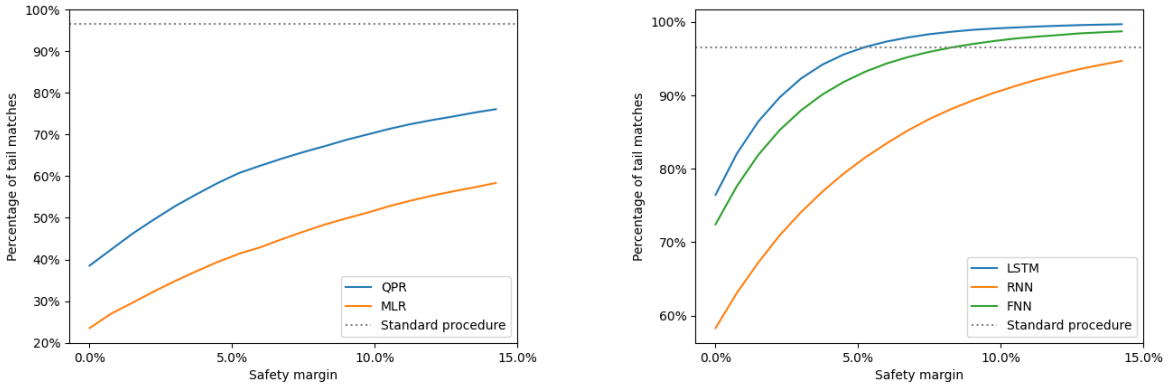
From a unique analytical standpoint, our numerical study offers a methodical exploration into the robustness of deep neural network models against noise in training labels. By employing the standard nested simulation procedure in Algorithm 1 as a data generator, we gain the ability to manipulate the 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 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.

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 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.

Figure 5 depicts the average percentage of correctly identified true tail scenarios[3] 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 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. For comparison, the horizontal line shows the true tail identification percentage for the standard nested simulation procedure. The

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

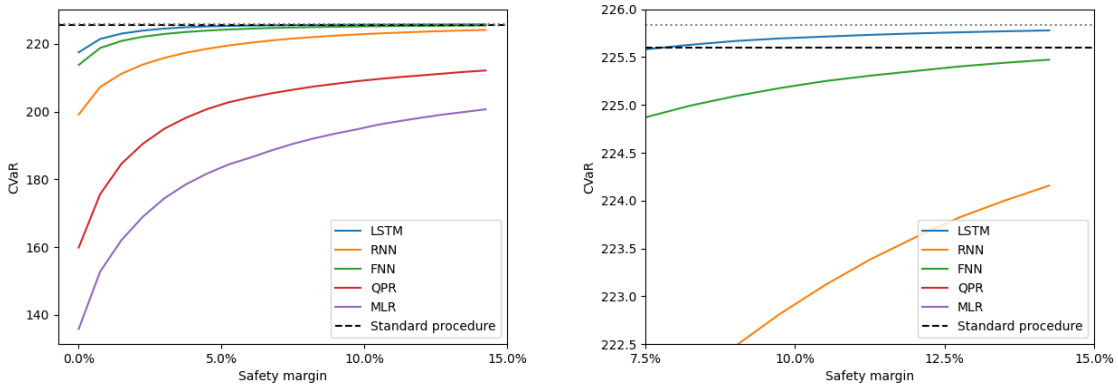


(a) Regression metamodels

(b) Neural network metamodel

Figure 5: Percentage of correctly identified true tail scenarios by different metamodels.

LSTM metamodel reaches similar percentage with 5% of safety margin, which corresponds to 5 times less simulation budget than the standard procedure's.



(a) Safety margin 0% - 15%.

(b) Safety margin 7.5% - 15%.

Figure 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 6 shows the average 95%-CVaR estimates for all five metamodels with different safety margins. Figure 6b is a zoomed version of Figure 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 6. We see that, with reasonable safety margins, the two-stage procedures with LSTM metamodels consistently produce estimates that are closer to the true value than the standard procedure's estimate. The LSTM metamodel is particularly superior as it accurately identifies the true tail scenarios and produces highly accurate CVaR estimates with small safety margins. For our two-stage procedure, achieving a similar accuracy as the standard procedure with a safety margin of 7.5% corresponds to a 77.5% reduction in the total simulation budget. 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.

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.

Model	**Noise level of training labels**	**Training error**	**True error**
Low-capcity LSTM	$N = 100$	0.075($\pm$0.004)	0.063($\pm$0.004)
High-capacity LSTM	$N = 100$	0.048($\pm$0.002)	0.060($\pm$0.002)
Low-capcity LSTM	$N = 10$	0.195($\pm$0.001)	0.070($\pm$0.001)
High-capacity LSTM	$N = 10$	0.157($\pm$0.002)	0.065($\pm$0.001)
Low-capcity LSTM	$N = 1$	1.366($\pm$0.006)	0.129($\pm$0.002)
High-capacity LSTM	$N = 1$	0.977($\pm$0.026)	0.378($\pm$0.017)

Table 2: Architectures and MSEs of LSTM metamodels.

Table 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 omitted since we are only interested in the metamodels' ability to generalize to the true noiseless feature-label relationship. The standard errors of these quantities are also reported in parentheses. We first observe that all the true errors are lower than the training errors, indicating that the LSTM metamodels generalize well to the true data. 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. This is an indication that the high-capacity LSTM metamodel is starting to overfit to the noise in the training labels. In contrast, the low-capacity LSTM metamodel is more robust to label noise and is able to learn the true feature-label relationship better from the high-noise dataset.

With more details, the QQ-plots depicted in Figure 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 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 low-capacity 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 low-capacity 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 low-capacity 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 low-capacity 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

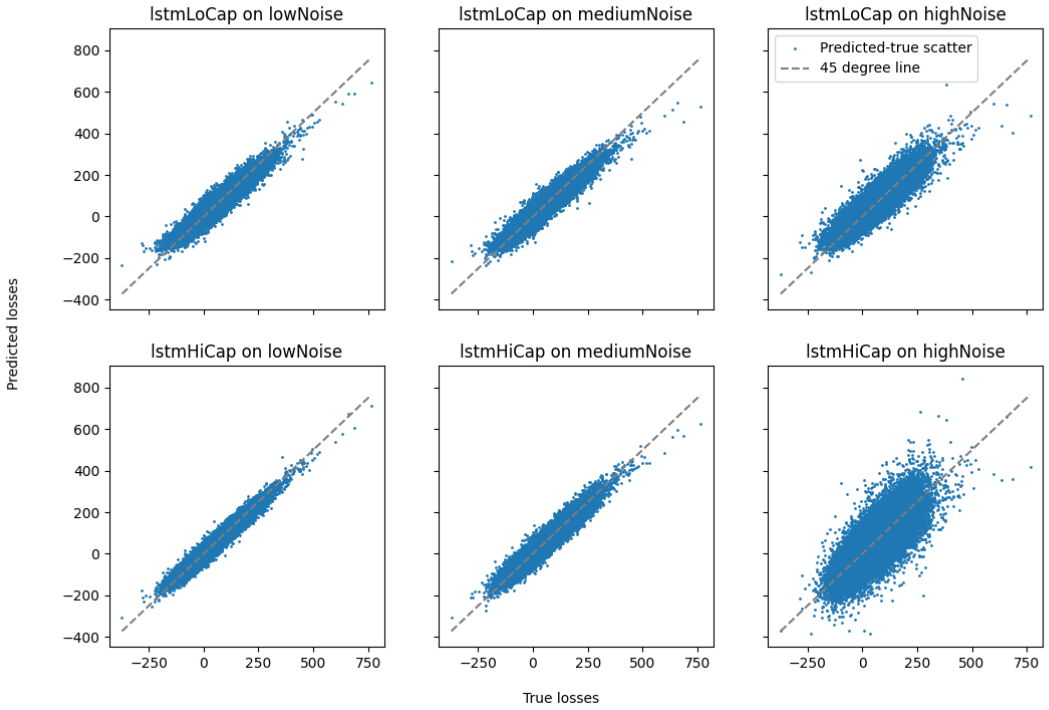


Figure 7: QQ-plots between true losses (x-axis) and predicted losses (y-axis) for two LSTM metamodels.

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: MSE between low-capacity LSTM predicted losses and true losses.

Table 3 shows the average squared errors between the low-capacity LSTM predictions and the true losses for various combinations of M and N. The 4th row of Table 3 shows the performance of the low-capacity LSTM metamodel trained with $M = 100{,}000$ samples with different noise levels. We observe that an increasing N reduces the MSE, but the reduction is not significant for N larger than 10. Another way to interpret the results in Table 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 suggest that the performance of the LSTM metamodel is more sensitive to the number of outer scenarios than the number of inner replications. Treating the neural network as an advanced regression metamodel, we find this pheonomenon to be consistent with the results in Broadie et al. (2015), where the authors show that the performance of a regression-based nested simulation procedure is more affected by the number of outer scenarios.

Table 4 reports the Spearman rank correlation coefficients that 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. It measures the metamodel's overall prediction accuracy. The p-values of all quantities are less than 10^{-5}, indicating that the correlations are statistically significant.

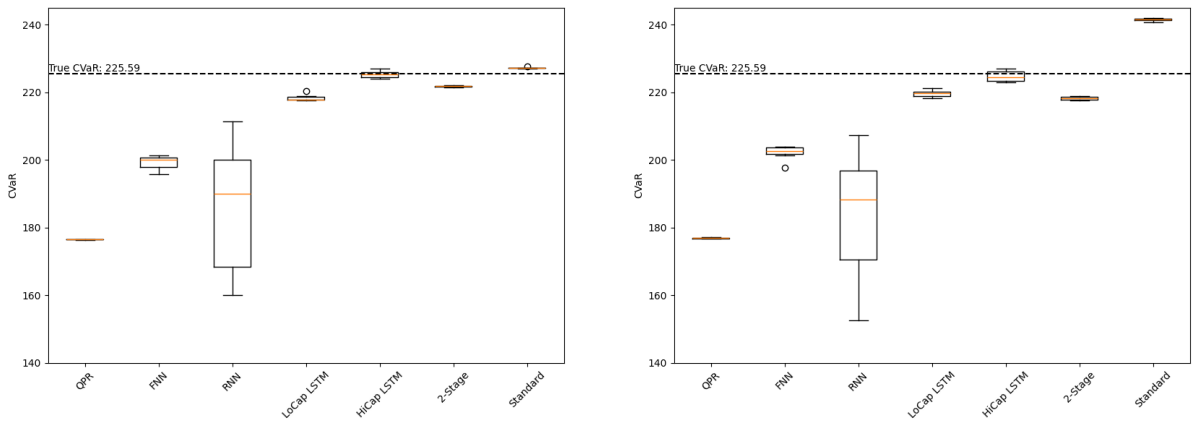
	$N = 1{,}000$	$N = 100$	$N = 10$	$N = 1$
$M = 100$	0.937 (0.941)	0.896 (0.903)	0.875 (0.915)	0.638 (0.881)
$M = 1{,}000$	0.922 (0.927)	0.886 (0.891)	0.899 (0.908)	0.722 (0.768)
$M = 10{,}000$	0.947 (0.948)	0.908 (0.905)	0.937 (0.900)	0.845 (0.630)
$M = 100{,}000$	0.965 (0.966)	0.927 (0.922)	0.963 (0.909)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

Table 4: Spearman (and Pearson) correlation coefficients between low-capacity LSTM predicted losses and true losses.

Consistent with the results in Table 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 low-capacity 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.

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, and we also examine the single-stage procedure's performance in estimating the standard deviation of the hedging losses. 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 5.1 is that the metamodels predictions are used to estimate the risk measures directly.



(a) With low noise training labels ($N = 100$)

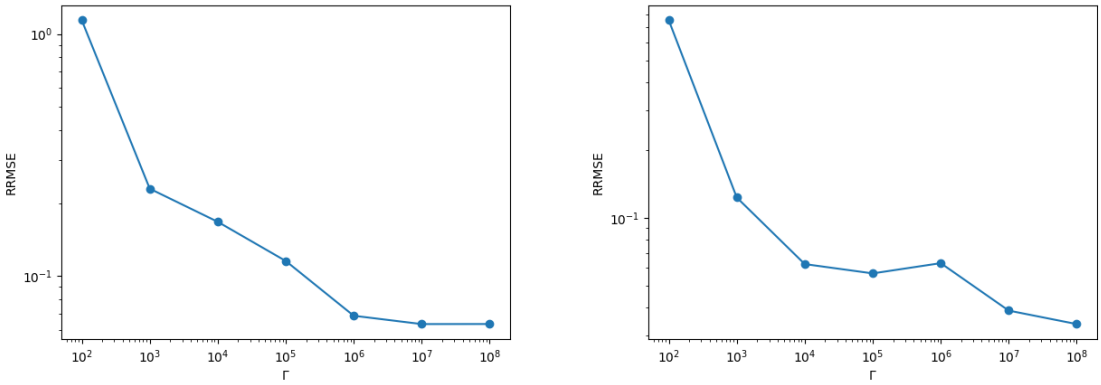
(b) With medium noise training labels ($N = 10$)

Figure 8: CVaR estimates of the single-stage procedure with metamodels.

Figure 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 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 single-stage procedure with the LSTM metamodels 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) low-capacity LSTM

(b) high-capacity LSTM

Figure 9: Empirical convergence of the single-stage procedure with LSTM metamodels.

Figure 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 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 ($N = 10$). Due to computational constraints, we are not able to run the two-stage procedure and the standard procedure with a higher M or N, 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 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 empirically in $\mathcal{O}(\Gamma^{-\frac{2}{3}})$. Hence, in

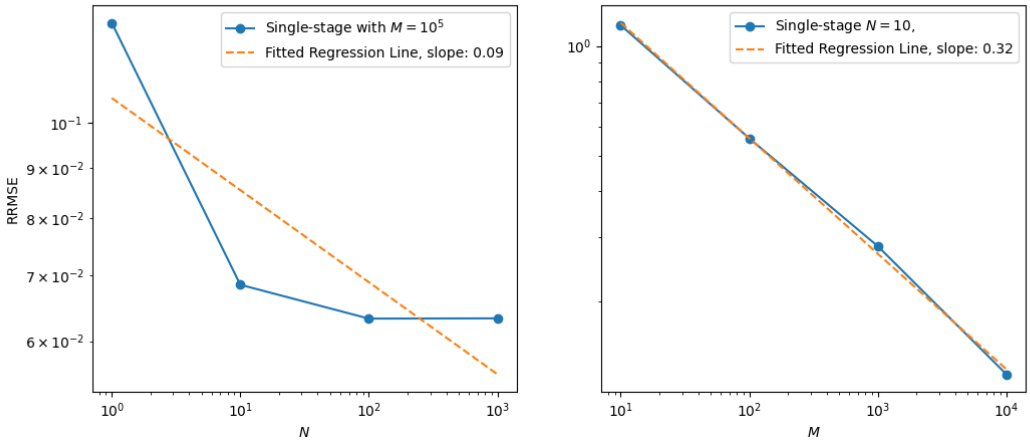


Figure 10: Empirical convergence of the single-stage procedure with a low-capacity LSTM metamodel.

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.

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 optimal 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.

References

Mark Broadie, Yiping Du, and Ciamac C Moallemi. Risk estimation via regression. *Operations Research*, 63(5):1077–1097, 2015.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

Mark J Cathcart, Hsiao Yen Lok, Alexander J McNeil, and Steven Morrison. Calculating variable annuity liability "greeks" using monte carlo simulation. *ASTIN Bulletin: The Journal of the IAA*, 45(2):239–266, 2015.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014. `https://arxiv.org/abs/1412.3555`. Accessed 7$^{\text{th}}$ Sep 2023.

Ou Dang. *Efficient Nested Simulation of Tail Risk Measures for Variable Annuities*. Ph.D. thesis, Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, ON, Canada, 2021. `https://uwspace.uwaterloo.ca/handle/10012/17084`.

Ou Dang, Mingbin Feng, and Mary R Hardy. Efficient nested simulation for conditional tail expectation of variable annuities. *North American Actuarial Journal*, 24(2):187–210, 2020.

EIOPA. The underlying assumptions in the standard formula for the solvency capital requirement calculation. Technical report, The European Insurance and Occupational Pensions Authority, 2014.

Daniel J Fonseca, Daniel O Navaresse, and Gary P Moynihan. Simulation metamodeling through artificial neural networks. *Engineering applications of artificial intelligence*, 16(3):177–183, 2003.

Guojun Gan and Sheldon X Lin. Valuation of large variable annuity portfolios under nested simulation: A functional data approach. *Insurance: Mathematics and Economics*, 62:138–150, 2015.

Paul Glasserman. *Monte Carlo Methods in Financial Engineering*, volume 53. Springer, 2004.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. `https://arxiv.org/abs/1412.6572`. Accessed 1$^{\text{st}}$ Jan 2024.

Mary R Hardy. *Investment Guarantees: Modeling and Risk Management for Equity-Linked Life Insurance*, volume 168. John Wiley & Sons, 2003.

Trevor Hastie, Robert Tibshirani, and Jerome H Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer, 2009.

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

Jeff L Hong, Sandeep Juneja, and Guangwu Liu. Kernel smoothing for nested estimation with application to portfolio risk measurement. *Operations Research*, 65(3):657–673, 2017.

Lu Jiang, Di Huang, Mason Liu, and Weilong Yang. Beyond synthetic noise: Deep learning on controlled noisy labels. In *International conference on machine learning*, pages 4804–4815. PMLR, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. `https://arxiv.org/pdf/1412.6980.pdf`. Accessed 7$^{\text{th}}$ Sep 2023.

Yann LeCun, Yoshua Bengio, and Geoffrey E Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Qui X Lieu, Khoa T Nguyen, Khanh D Dang, Seunghye Lee, Joowon Kang, and Jaehong Lee. An adaptive surrogate model to structural reliability analysis using deep neural network. *Expert Systems with Applications*, 189:116104, 2022.

Ming Liu and Jeremy Staum. Stochastic kriging for efficient nested simulation of expected shortfall. *Journal of Risk*, 12 (3):3, 2010.

Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 29, 2016.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015. `https://arxiv.org/abs/1511.06807`. Accessed 1$^{\text{st}}$ Jan 2024.

OpenAI. Chatgpt, 2023. `https://chat.openai.com/chat`. Accessed 7$^{\text{th}}$ Sep 2023.

OSFI. Life insurance capital adequacy test. Technical report, Office of the Superintendent of Financial Institutes Canada, 2017.

Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli. Analyzing noise in autoencoders and deep networks. *arXiv preprint arXiv:1406.1831*, 2014. `https://arxiv.org/abs/1406.1831`. Accessed 1$^{\text{st}}$ Jan 2024.

R Tyrrell Rockafellar and Stanislav Uryasev. Conditional value-at-risk for general loss distributions. *Journal of Banking & Finance*, 26(7):1443–1471, 2002.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California University San Diego La Jolla Institute for Cognitive Science, 1985.

Isabelle Salle and Murat Yıldızoğlu. Efficient eampling and meta-modeling for computational economic models. *Computational Economics*, 44:507–536, 2014.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. https://arxiv.org/abs/1312.6199. Accessed 1st Jan 2024.

The Geneva Association. Variable annuities—an analysis of financial stability. Technical report, 2013. https://www.genevaassociation.org/sites/default/files/research-topics-document-type/pdf_public/ga2013-variable_annuities_0 Accessed 7th Sep 2023.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Kun Zhang, Mingbin Ben Feng, Guangwu Liu, and Shiyu Wang. Sample recycling for nested simulation with application in portfolio risk measurement. 2022. https://arxiv.org/abs/2203.15929. Accessed 2nd May 2023.