Xintong Li
xintong.li1@uwaterloo.ca

Dept. Statistics and Actuarial Science
University of Waterloo

Efficient Machine Learning Approaches for Fast Risk Evaluation of VAs

Supervised by Prof. Tony Wirjanto and Prof. Mingbin Feng

Thesis Defense, University of Waterloo

Outline

Nested Simulation Procedures

Nested simulation procedures are necessary for **complex** financial derivatives and insurance products.

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

Involves two levels of Monte Carlo simulations:

- Outer level: generates underlying risk factors (outer scenarios), $X_i \sim F_X$.
- Inner level: generates scenario-wise samples of portfolio losses (inner replications), $Y_{ij} \sim F_{Y|X_i}$.
- Total simulation budget: $\Gamma = M \cdot N$.

Computationally expensive due to its nested structure.

Common Risk Measures

- Smooth h, e.g., quadratic tracking error,

$$\rho(L) = \mathbb{E}\left[(L-b)^2\right];$$

- hockey-stick h: mean excess loss,

$$\rho(L) = \mathbb{E}\left[L \cdot \mathbb{1}_{\{L \geq u\}}\right];$$

- indicator h: probability of large loss,

$$\rho(L) = \mathbb{E}\left[\mathbb{1}_{\{L \geq u\}}\right];$$

- Value at Risk (VaR),

$$\rho_\alpha(L) = Q_\alpha(L) = \inf\{u : \mathbb{P}(L \leq u) \geq \alpha\};$$

- Conditional Value at Risk (CVaR) [1],

$$\rho_\alpha(L) = \mathbb{E}\left[L | L \geq Q_\alpha(L)\right].$$

[1] Note: If $Q_\alpha(L)$ falls in a probability mass, $\rho(L) = \frac{(\beta-\alpha)Q_\alpha(L)+(1-\beta)\mathbb{E}[L|L\geq Q_\alpha(L)]}{1-\alpha}$.

Standard Nested Simulation

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

- Uses inner sample mean to estimate $L(X_i)$.
- Proposed by Gordy and Juneja (2010); finds optimal growth order of M and N.
- Zhang et al. (2021) estimate the optimal M and N using a bootstrap method.
- Computationally expensive and potentially **wasteful** use of budget.

Other Nested Simulation Procedures

Subsequent works focus on improving the efficiency of nested simulation:

- Regression-based (Broadie et al., 2015)
- Kernel smoothing (Hong et al., 2017)
- Likelihood ratio (Feng and Song, 2020)
- Kernel ridge regression (Zhang et al., 2022)

Key ideas:

- Pool inner replications from different outer scenarios
- Use metamodeling techniques to approximate the inner simulation model

Metamodeling Approach

In this thesis, we focus on procedures that use **supervised learning metamodels** to approximate the inner simulation model.

- Treat the inner simulation as a black-box function
- Approximate $L(\cdot)$ with $\hat{L}^{\mathsf{SL}}_{M,N}(\cdot)$
- Train with a set of feature-label pairs generated from the standard procedure:

$$\{(X_i, \hat{L}_{N,i}) | i = 1, \dots, M, j = 1, \dots, N\}$$

- Use trained metamodel to make predictions for all $X \in \mathcal{X}$

There are **computational costs** associated with pooling inner replications.

Problem Statement

Minimize mean squared error (MSE) of the estimator subject to total simulation budget:

$$\min_{M,N} \quad \mathbb{E}\left[\left(\hat{\rho}_{M,N} - \rho\right)^2\right]$$

$$\text{subject to } M \cdot N = \Gamma$$

Interested in convergence order as $\Gamma \to \infty$

Asymptotic Convergence Rates of Different Procedures

Procedures	**Smooth** h	**Hockey-Stick** h	**Indicator** h
Standard Procedure	$\mathcal{O}(\Gamma^{-2/3})$	$\mathcal{O}(\Gamma^{-2/3})$	$\mathcal{O}(\Gamma^{-2/3})$
Regression	$\mathcal{O}(\Gamma^{-1})$	$\mathcal{O}(\Gamma^{-1+\delta})$	No Result
Kernel Smoothing	$\mathcal{O}(\Gamma^{-\min(1,4/(d+2))})$		
Kernel Ridge Regression	$\mathcal{O}(\Gamma^{-1})$		
Likelihood Ratio	$\mathcal{O}(\Gamma^{-1})$		

- We show the asymptotic convergence rates of the standard procedure for smooth and hockey-stick h.
- Only kernel smoothing depends on the asset dimension d.

Key Theoretical Results

Observations:

- Most literature focuses on the MSE of $\hat{\rho}$.
- Wang et al. (2022) analyze convergence of absolute error in probabilistic order.

Contribution: bridging the gap between MSE and absolute error convergence.

- Convergence in MSE:

$$\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right] = \mathcal{O}\left(\Gamma^{-\xi}\right)$$

- Convergence in Probabilistic Order:

$$|\hat{\rho}_\Gamma - \rho| = \mathcal{O}_{\mathbb{P}}(\Gamma^{-\xi})$$

Key Theoretical Results

Theorem

If $\hat{\rho}_\Gamma$ converges in MSE to ρ in order ξ, then $\hat{\rho}_\Gamma$ converges in probabilistic order to ρ in order $\frac{\xi}{2}$.

- First result to draw connection between MSE and probabilistic order convergence.
- Applicable to any nested simulation procedure.
- Convergence in MSE implies convergence in probabilistic order.

Experiment Design

We compare 5 nested simulation procedures

- Standard nested simulation
- Regression-based
- Kernel smoothing
- Likelihood ratio
- Kernel ridge regression

And their empirical convergence stable across different:

- Risk measures
- Option types
- Asset dimensions
- Asset models (GBM vs. Heston)
- Regression bases (only for the regression-based procedure)

Finite-Sample Performance

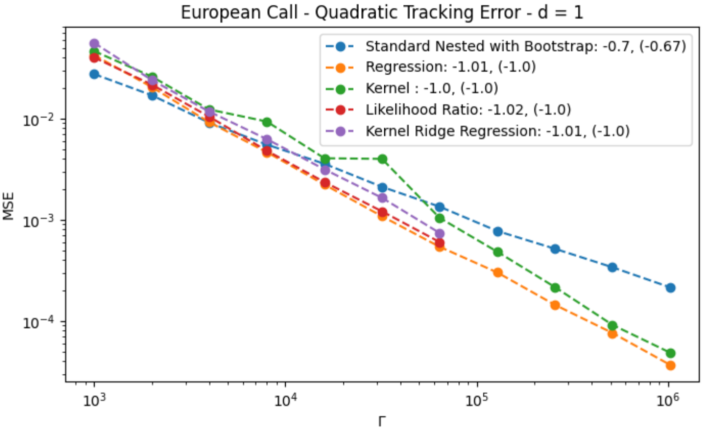
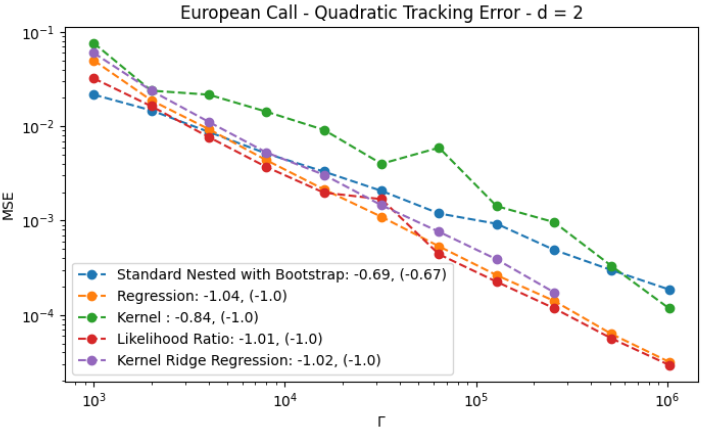


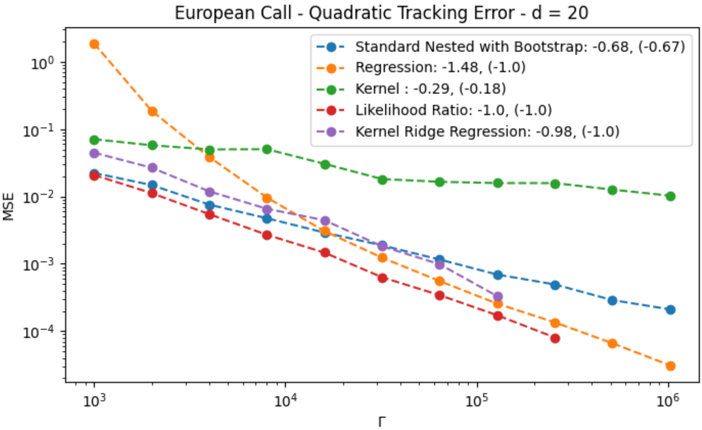
Figure: Empirical convergence rates of different procedures for the base case

Sensitivity to Asset Dimension



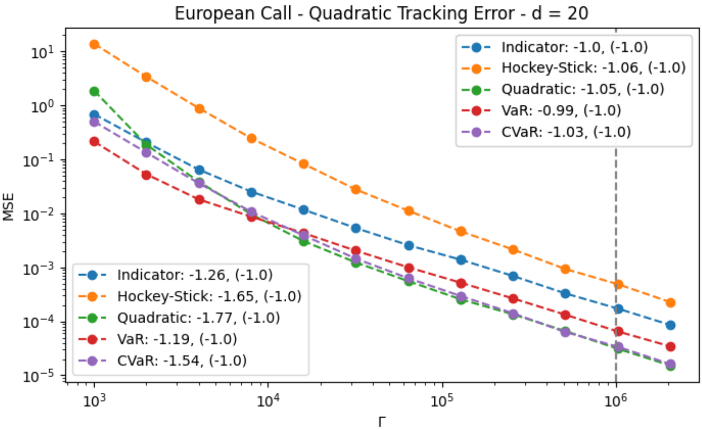
- Standard, KRR, and likelihood ratio procedures are dimension-independent
- Kernel smoothing and regression show sensitivity to dimension, but in different ways

Sensitivity to Asset Dimension



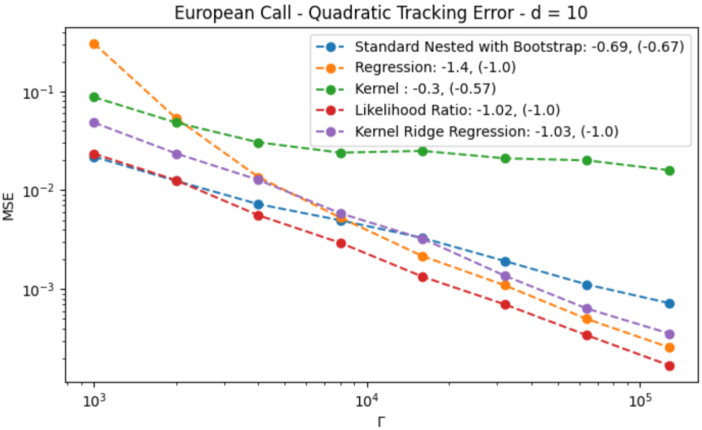
- Standard, KRR, and likelihood ratio procedures are dimension-independent
- Kernel smoothing and regression show sensitivity to dimension, but in different ways

Fast Convergence of Regression-based Procedure



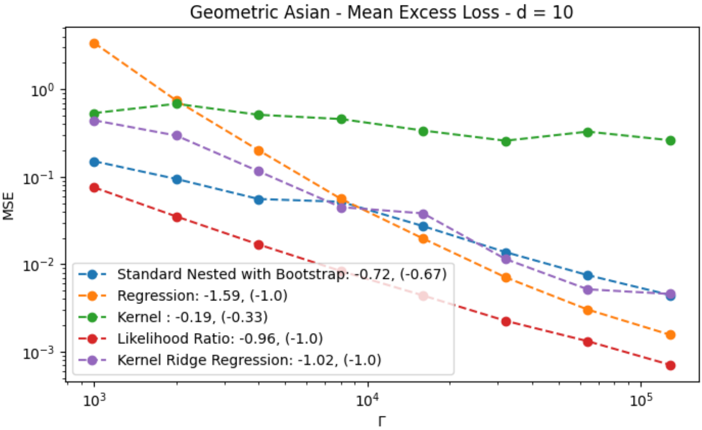
- Higher initial convergence rate
- Stabilizes to match asymptotic rate at higher budgets
- Consistent across different asset dimensions

Sensitivity to Option Type



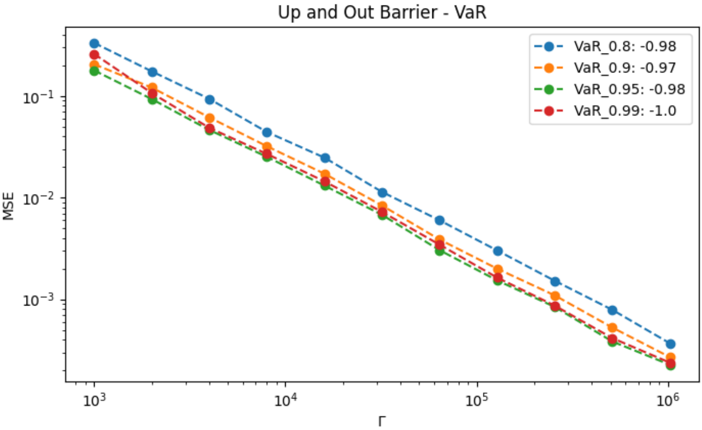
- Similar convergence patterns across different option types
- Regression and kernel smoothing show higher empirical rates for barrier options

Sensitivity to Risk Measure



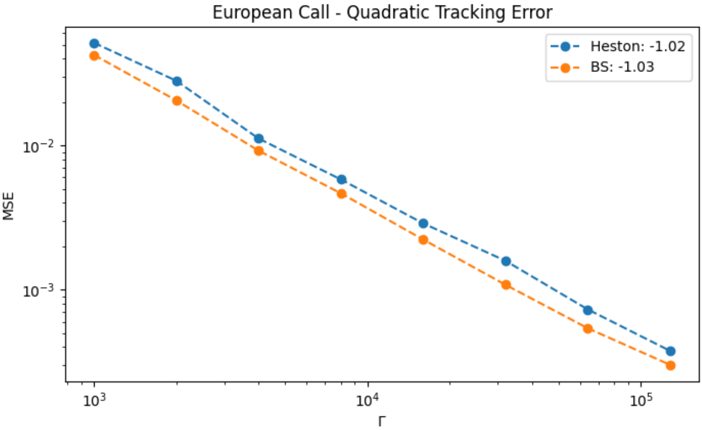
- Convergence behavior consistent across different risk measures
- Regression-based method shows highest empirical convergence rates

Sensitivity to VaR/CVaR Level



- Regression-based method not sensitive to VaR/CVaR level
- Consistent performance across different levels

Sensitivity to Asset Model



- Regression-based method insensitive to asset model (GBM vs. Heston)
- Consistent performance across different asset models

Computational Complexity

There are **computational costs** associated with pooling inner replications.

- Standard procedure: cost of estimating the optimal M and N
- Regression: most efficient among metamodel-based procedures
- Kernel smoothing: costly distance calculations and cross-validation
- Likelihood ratio: No training, but costly weight calculations
- KRR: even more expensive than kernel smoothing

Total Computation Time

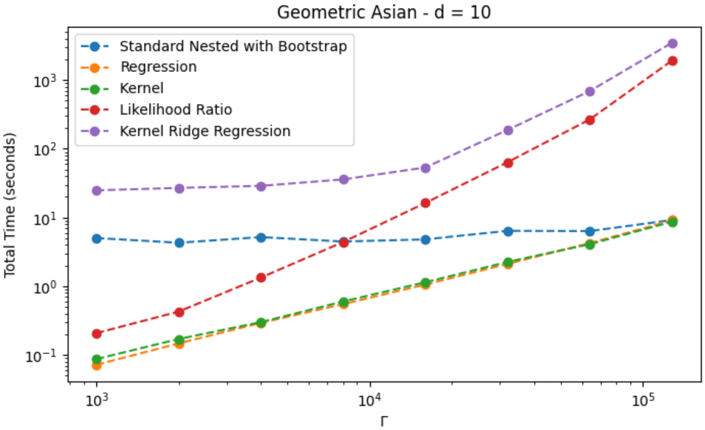


Figure: Total computation time for different procedures

Cost of Hyperparameter Tuning

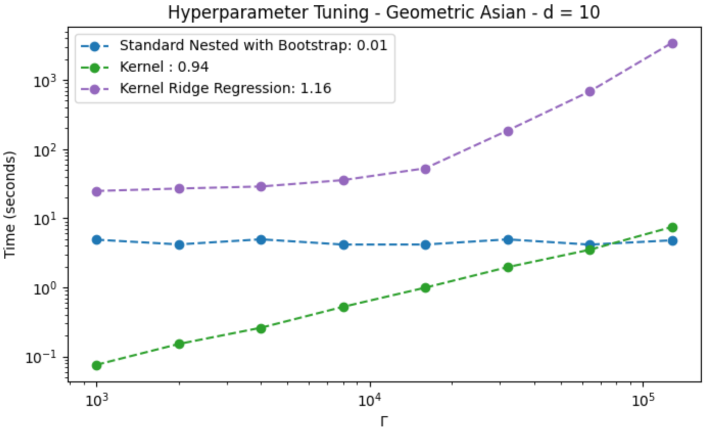


Figure: Cost of hyperparameter tuning for different procedures

Cost of Model Fitting and Validation

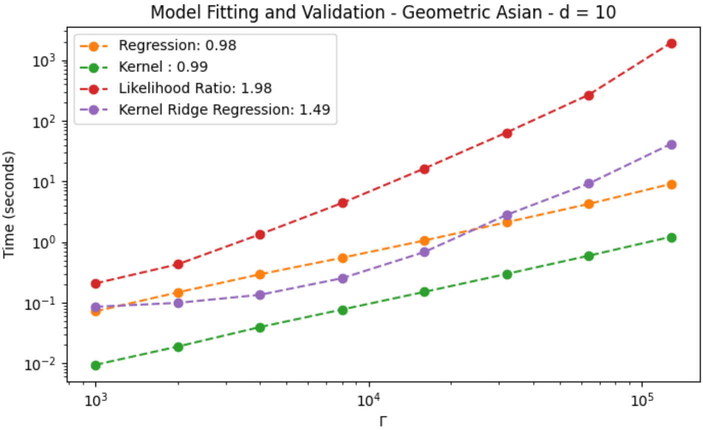


Figure: Cost of model fitting and validation for different procedures

Conclusion

Regression-based nested simulation procedure:

- Most robust and stable for limited budgets
- Efficient to implement
- Fast empirical convergence for option portfolios

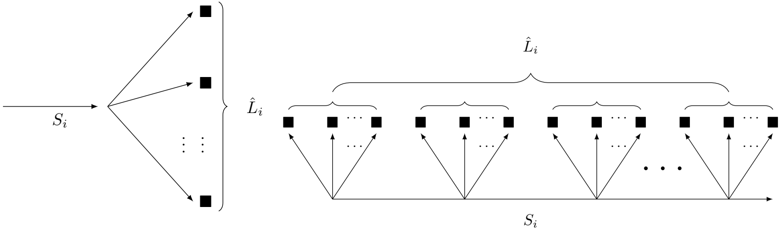
For high-dimensional or complex payoffs:

- Difficult to find a good regression basis
- Neural network-based procedures may be more suitable

Next project: examining performance of metamodel-based simulation procedures for variable annuities

From Options to Variable Annuities

Variable annuities (VAs) poses a challenge for nested simulation due to its **high-dimensional** and **complex payoff structure**.



1 Outer Path for Options

1 Outer Path for VAs

- Need to reconstruct a metamodeling-based nested simulation procedure

Nested Simulation for Risk Management of VAs

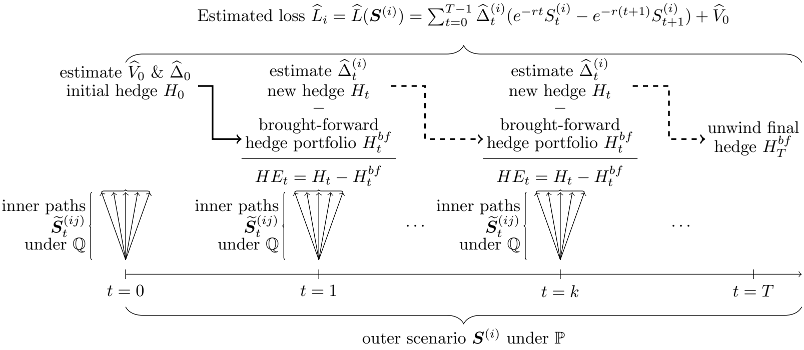


Figure: Illustration of nested simulation that estimates the P&L for one outer scenario

Standard Nested Simulation for VAs

Standard nested simulation for VAs is similar to the one for options.

- Generate M outer scenarios
- For each outer scenario,
  - Perform N inner simulations
  - Estimate hedging loss L_i with $\hat{L}_i$
- Use estimated losses to calculate tail risk measures (e.g., 95%-CVaR)

Observations:

- computational budget is limited;
- high-dimensional input space;
- only a **small** portion of scenarios are relevant when estimating **tail** risk measures.

Metamodel-based Nested Simulation

We use deep neural networks (DNNs) as metamodels

- Use LSTMs for sequential data
- **Challenge**: lack of transparency and interpretability

Research Contributions:

1. Propose two generic DNN-based nested simulation procedures
   - Accurate tail scenario identification
   - Significant computational savings by **budget concentration**
2. Study noise tolerance of DNNs using simulated data
   - **Control noise levels** by adjusting simulation parameters
   - Provide direct evidence on transparency and interpretability

Two-Stage Metamodel-based Nested Simulation

Algorithm Two-Stage Metamodel-based Nested Simulation for VAs

1: **Generate training data for metamodels:**

- Use a fraction of the simulation budget to run the standard nested simulation procedure with M outer scenarios and N' inner replications.
- Construct feature-label pairs $\{(X_i, Y_{ij}) : i = 1, \ldots, M, j = 1, \ldots, N'\}$

2: **Train metamodels:**

- Use the feature-label pairs to train a metamodel.
- Use the trained metamodel to make predictions for $\{X_i : i = 1, \ldots, M\}$.
- Sort the predicted losses to identify a predicted tail scenario set that contains the m largest predicted losses.

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

- Run the standard procedure on the predicted tail scenarios.
- Estimate the α-CVaR of L using the estimated losses on the predicted tail scenarios.

Benefits of a Two-Stage Procedure

Simulation budget can be saved when:

- the metamodel is accurate (a small m includes most tail scenarios)
- the metamodel can tolerate noise in training labels (a small N')

Key findings:

- Substantial computational savings ($70\% - 85\%$ reduction)
- Maintains accuracy comparable to standard procedure
- DNN metamodels can distinguish between tail and non-tail scenarios effectively
- Addresses regulatory concerns by using actual simulations for final estimates

Another finding: some DNN metamodels make **accurate loss predictions** for given scenarios.

Single-Stage Metamodel-based Nested Simulation

Algorithm Single-Stage Metamodel-based Nested Simulation for VAs

1: **Generate training data for metamodels:**
- Use the entire simulation budget to run the standard nested simulation procedure with M outer scenarios and N inner replications.
- Construct feature-label pairs $\{(X_i, Y_{ij}) : i = 1, \ldots, M, j = 1, \ldots, N\}$

2: **Train metamodels:**
- Use the feature-label pairs to train a metamodel.
- Use the trained metamodel to make predictions for $\{X_i : i = 1, \ldots, M\}$.

3: **Use metamodel predictions to estimate tail risk measures directly.**

Key advantages:
- more efficient than a two-stage procedure;
- avoids specifying m.

Experiment Setting

We estimate the 95%-CVaR of the hedging loss for a GMWB contract with 20-year maturity.

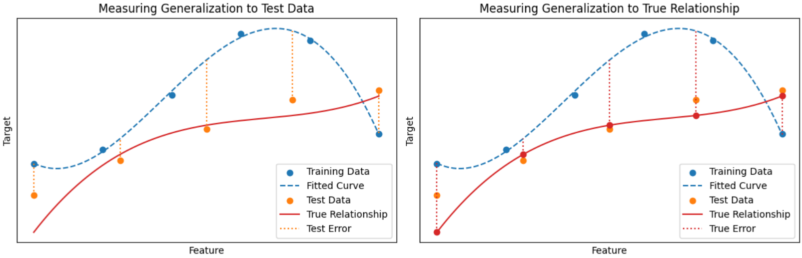
Specifications:

- The underlying asset follows a regime-switching geometric Brownian motion;
- The contract is delta-hedged monthly (240 periods);
- The true 95%-CVaR is estimated using 100,000 outer scenarios and 100,000 inner replications.
- The metamodel is trained using 90,000 outer scenarios and 100 inner replications.
- Benchmark: standard nested simulation procedure with 100,000 outer scenarios and 1,000 inner replications.

Experiment Design

Research Questions:

- What do DNNs learn from noisy data?
- How well do DNNs learn from noisy data?



- Our 90,000 training data is noisy, and the test data is also **noisy**.
- Our evaluation is based on the true (**noiseless**) feature-label relationship[2].

[2] Made possible by novel simulation design.

Experiment Setting

We consider the following metamodel architectures:

Metamodel	Abbreviation	Capacity
Multiple Linear Regression	MLR	241
Quadratic Polynomial Regression	QPR	481
Feedforward Neural Network	FNN	35,009
Recurrent Neural Network	RNN	32,021
Long Short-Term Memory	LSTM	35,729

Table: Metamodel architectures for GMWB inner simulation model

Capacity is defined as the number of parameters in the metamodel.

- Higher capacity metamodels are more flexible and expressive.
- Lower capacity metamodels are less likely to overfit.

Traditional Regression Metamodels

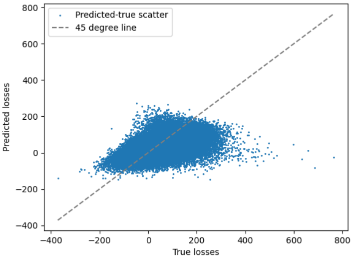


Figure: QQ plots between true and predicted loss labels for MLR and QPR metamodels

- MLR and QPR metamodels make **inaccurate** loss predictions.
- Feature engineering is hardly feasible for our 240-dimensional X.

Deep Neural Network Metamodels

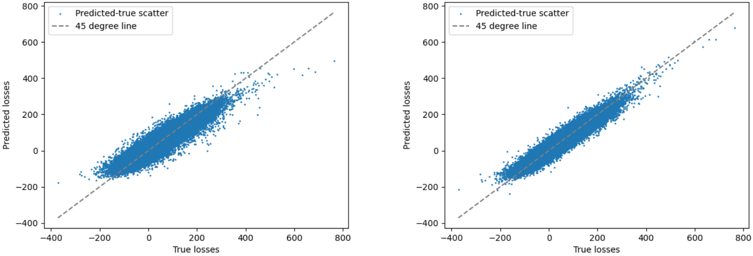


Figure: QQ plots between true and predicted loss labelsfor FNN and LSTM metamodels

- DNN metamodels are more flexible.
- Time series features prefer a LSTM metamodel over FNN.
- Network architecture serves as prior knowledge that regularizes DNNs.

Metamodel Performance on Different Datasets

Metamodel	Training error	Test error	True error
MLR	$0.706(\pm 8.3 \times 10^{-4})$	$0.713(\pm 2.7 \times 10^{-2})$	$0.706(\pm 3.4 \times 10^{-4})$
QPR	$0.543(\pm 8.3 \times 10^{-4})$	$0.554(\pm 2.7 \times 10^{-2})$	$0.544(\pm 4.1 \times 10^{-4})$
FNN	$0.129(\pm 6.0 \times 10^{-3})$	$0.240(\pm 9.8 \times 10^{-3})$	$0.132(\pm 5.8 \times 10^{-3})$
RNN	$0.132(\pm 7.5 \times 10^{-3})$	$0.137(\pm 7.6 \times 10^{-3})$	$0.119(\pm 7.5 \times 10^{-3})$
LSTM	$0.075(\pm 4.5 \times 10^{-3})$	$0.079(\pm 5.4 \times 10^{-3})$	$0.063(\pm 4.4 \times 10^{-3})$
RNN*[3]	$0.109(\pm 5.2 \times 10^{-3})$	$0.128(\pm 5.2 \times 10^{-3})$	$0.109(\pm 5.2 \times 10^{-3})$

Table: Average MSEs and 95% confidence bands of metamodels for GMWB inner simulation model.

- RNN-based metamodels have lower true errors than their training errors.
- DNN metamodels with suitable architectures **cut through the noise** in training labels.

[3] This row summarizes the results of the well-trained RNNs.

Issues with RNNs

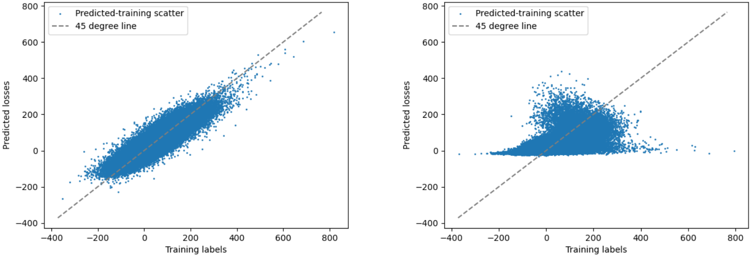


Figure: QQ plots between training and predicted loss labels for RNN metamodels

- RNN metamodels suffers from **vanishing gradient problem**.
- Ease of training (reliability) is a critical factor when choosing a DNN metamodel.

Safety Margin

Consider estimating the 95% CVaR with 100,000 outer scenarios.

Figure: Illustration: a safety margin of 4% ($m = 9000$)

Choosing a safety margin: a trade-off between accuracy and efficiency.

- A lower margin: not enough tail identified.
- A higher margin: more accurate CVaR estimate, but more budget needed to perform extensive inner simulations.

Tail Scenario Identification

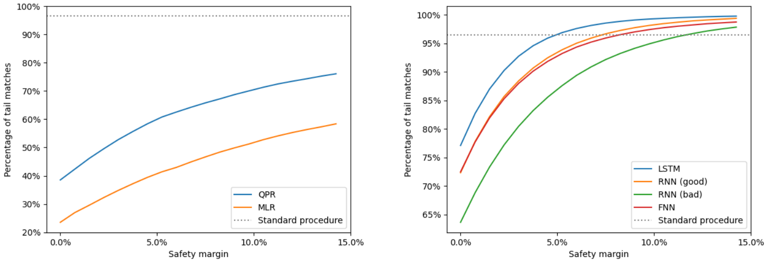


Figure: Tail scenario identification for regression and DNN metamodels

- Traditional regression metamodels are **unable** to accurately identify tail scenarios even with high safety margins.
- LSTM metamodels surpasses the standard procedure with 5% safety margin.

Estimating CVaR

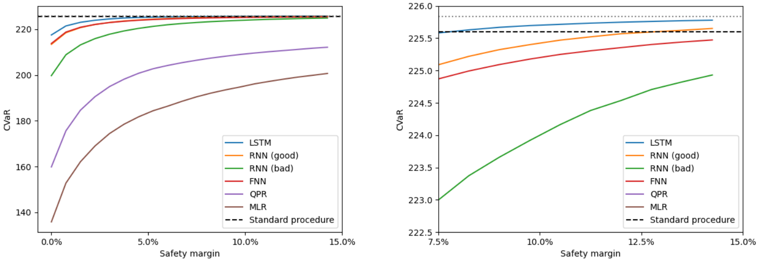


Figure: CVaR estimation for DNN metamodels

- Traditional regression metamodels are **unable** to accurately estimate CVaR even with high safety margins.
- LSTM surpasses the standard procedure with a **5% safety margin**.
- With a 95% safety margin, any two-stage procedure produce the same CVaR estimate as a standard procedure.

Sensitivity Testing for DNNs

In a simulation study, we have control over the **noise level in training labels** and the **number of training samples**.

Controling the inner replications N' varies the noise level in training labels.

- **Low noise labels**: $N' = 100$
- **Medium noise labels**: $N' = 10$
- **High noise labels**: $N' = 1$

Controling the outer scenarios M varies the number of training samples.

- $M \in \{10^2, 10^3, 10^4, 10^5\}$

2 LSTMs of **different capacities** are examined based on their MSEs.

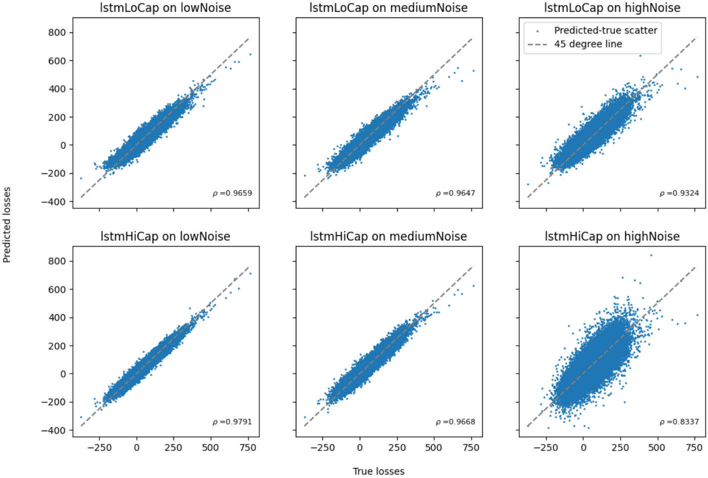
Noise Tolerance of DNNs

Model	N'	Training error	Test error	True error
LSTM	100	0.075	0.079	0.063
High-capacity LSTM	100	0.068	0.102	0.060
Average Difference	100	−0.007	0.023	−0.003
LSTM	10	0.195	0.193	0.070
High-capacity LSTM	10	0.157	0.199	0.065
Average Difference	10	−0.038	0.006	−0.005
LSTM	1	1.366	0.781	0.129
High-capacity LSTM	1	1.354	0.795	0.149
Average Difference	1	−0.012	0.014	0.020

Table: MSEs of LSTM metamodels.

- Both LSTMs cut through the noise in training labels.
- Both LSTMs deteriorate dramatically on **high-noise** labels.
- High-capacity LSTM can tolerate **low** and **medium** label noise.

Noise Tolerance of DNNs



Sensitivity of Regular LSTM

	$N' = 1$	$N' = 10$	$N' = 100$	$N' = 1000$
$M = 100$	1.139	0.229	0.167	0.158
$M = 1000$	0.559	0.173	0.123	0.127
$M = 10000$	0.283	0.115	0.099	0.097
$M = 100000$	0.129	0.070	0.063	0.063

Table: MSE between regular LSTM's predicted losses and true losses.

- Same color $\rightarrow$ same total simulation budget.
- $N' = 10$ is a reasonable budget allocation for LSTM metamodels.

Sensitivity of High-capacity LSTM

	$N' = 1$	$N' = 10$	$N' = 100$	$N' = 1000$
$M = 100$	0.764	0.408	0.131	0.087
$M = 1000$	0.878	0.367	0.156	0.087
$M = 10000$	0.351	0.147	0.064	0.063
$M = 100000$	0.149	0.065	0.060	0.038

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

- Same color $\rightarrow$ same total simulation budget.
- $N' = 10$ is a reasonable budget allocation for LSTM metamodels.

Single-Stage Procedure

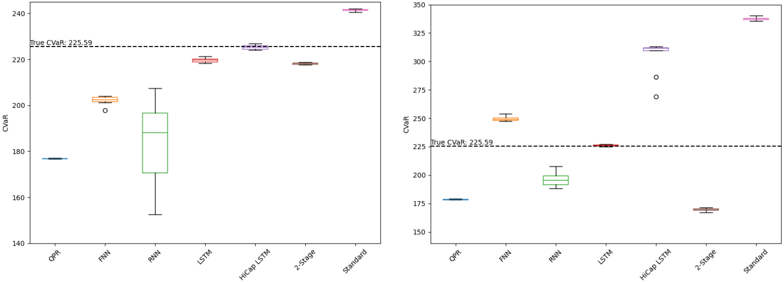


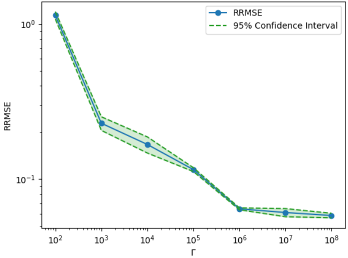
Figure: CVaR estimates of single-stage procedures (left: $N' = 10$. right: $N' = 1$).

- The single-stage procedure outperforms the two-stage procedure.
- The single-stage procedure is more efficient than the two-stage procedure.
- Setting $N' = 10$ is a reasonable budget allocation.

Convergence Analysis

For each Γ, the best performing metamodel is used.

- Maximum number of outer scenarios $M = 10^5$.



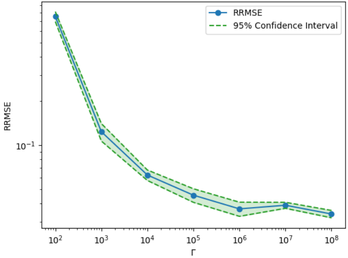


Figure: Empirical convergence of CVaR for single-stage procedures with LSTM metamodels (left: regular LSTM. right: high-capacity LSTM).

- Minimal effect of increasing N' on CVaR estimation.
- Similar behavior as regression metamodels ($d = 20$) in the previous section.

Convergence Analysis

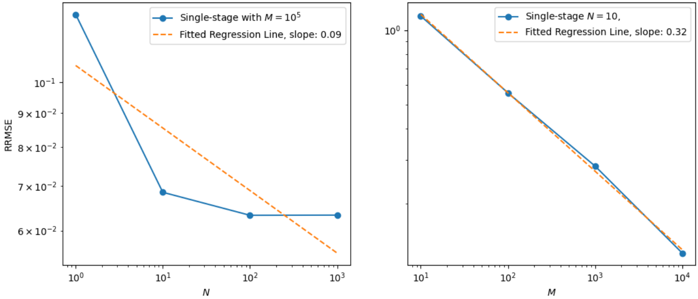


Figure: Empirical convergence of the single-stage procedure with a LSTM metamodel.

- Minimal effect of increasing N' on CVaR estimation.
- For a given Γ, set N' constant and allocate budget to outer simulations.

Conclusion

Key Findings:

- LSTMs are **resilient** to moderate levels of noise in training labels.
- Deep neural networks can learn **true** complex dynamic hedging model.
- Two-stage procedure addresses regulatory concerns by avoiding direct use of metamodel predictions.
- Single-stage procedure is **efficient** and **versatile**.
- **Increasing outer scenarios** is more beneficial.
- High-capacity LSTM requires lower noise training labels.

Future Directions:

- Apply deep neural network metamodels to other risk management tasks.
- Investigate impact of label noise on other deep learning models
- Explore optimal network architectures for different simulation models.

Transfer Learning for Rapid Adaptation of DNN Metamodels

Challenge: Adapting deep neural network metamodels to changing conditions.

- Retraining from scratch is computationally expensive.
- Efficient incorporation of new VA contract data.
- Balancing model accuracy and computational costs.

Solution: Transfer learning (TL) to develop adaptable, efficient metamodels for VA dynamic hedging.

- Pre-train deep neural network on contracts with abundant simulation data.
- Fine-tune on smaller dataset of new contracts/market conditions.
- Leverages shared features between VA contracts.
- Computational savings:
  - Reduced training time.
  - Fewer data points needed for good performance.

Transfer Learning Framework

Key Components:

- **Domain** $\mathcal{D}$: feature space $\mathcal{X}$ + probability distribution F
- **Task** $\mathcal{T}$: label space $\mathcal{Y}$ + predictive function $f : \mathcal{X} \to \mathcal{Y}$

Source vs. Target:

- **Source:** $\mathcal{D}_{\text{So}} = \{\mathcal{X}_{\text{So}}, F_{\text{So}}(X)\}$
- **Target:** $\mathcal{D}_{\text{Ta}} = \{\mathcal{X}_{\text{Ta}}, F_{\text{Ta}}(X)\}$

Our Goal:

- **Input features** X: risk factors from outer simulation
- **Output labels** L: contract losses
- **Source and target**: from VAs with abundant simulation data to new VAs with limited data
- **Goal**: improve $f_{\text{Ta}}(\cdot)$ using knowledge from $\mathcal{D}_{\text{So}}$ and $f_{\text{So}}(\cdot)$

Transfer Learning Techniques

Common Techniques:

- **Fine-tuning:** a model pre-trained on a source task is used as a starting point for a target task.
- **Layer freezing:** only part of the model is fine-tuned.
- **Multi-task learning:** perform training on multiple tasks simultaneously.

Key considerations:

- Similarity between source and target tasks
- Appropriate learning rate

Fine-tuning Algorithm for LSTM Metamodels in VA Hedging

Algorithm Fine-tuning Algorithm for LSTM Metamodels in VA Hedging

1: **Input:** $\mathcal{D}_{\mathsf{So}} = \{(X_{\mathsf{So}}^{(i)}, L_{\mathsf{So}}^{(i)})\}_{i=1}^{M_{\mathsf{So}}}$, $\mathcal{D}_{\mathsf{Ta}} = \{(X_{\mathsf{Ta}}^{(i)}, L_{\mathsf{Ta}}^{(i)})\}_{i=1}^{M_{\mathsf{Ta}}}$, α_{So}, and α_{Ta}.

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

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

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

$$\theta_{\mathsf{Ta}} \leftarrow \theta_{\mathsf{So}}$$

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

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

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

Layer Freezing Algorithm for LSTM Metamodels in VA Hedging

Algorithm Layer Freezing Algorithm for LSTM Metamodels in VA Hedging

1: **Input:** $\mathcal{D}_{\mathsf{So}} = \{(X_{\mathsf{So}}^{(i)}, L_{\mathsf{So}}^{(i)})\}_{i=1}^{M_{\mathsf{So}}}$, $\mathcal{D}_{\mathsf{Ta}} = \{(X_{\mathsf{Ta}}^{(i)}, L_{\mathsf{Ta}}^{(i)})\}_{i=1}^{M_{\mathsf{Ta}}}$, α_{So}, and α_{Ta}.

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

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

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

4: Freeze the parameters of the shared layers θ_0 and fine-tune the trainable layers θ_1 on the target dataset $\mathcal{D}_{\mathsf{Ta}}$:

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

5: **Output:** Adapted model $f_{\mathsf{Ta}}(\cdot;\theta_{\mathsf{Ta}})$ for the target task.

Multi-task Learning Algorithm for LSTM Metamodels in VA Hedging

Algorithm Multi-task Learning Algorithm for LSTM Metamodels in VA Hedging

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

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

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

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

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

Experiment Setup

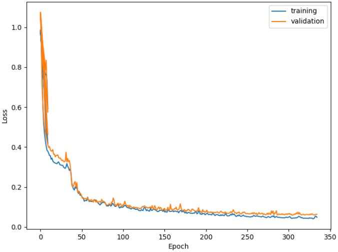
Contract	Asset Model	Lapse	M_{So}	M_{Ta}
GMMB	GBM	No lapse	50000	N/A
GMMB	RS-GBM	No lapse	50000	2000
GMMB	RS-GBM	Static lapse	50000	2000
GMMB	RS-GBM	Dynamic lapse	50000	2000
GMWB	RS-GBM	Dynamic lapse	N/A	2000

Table: VA Contracts for Transfer Learning Experiments

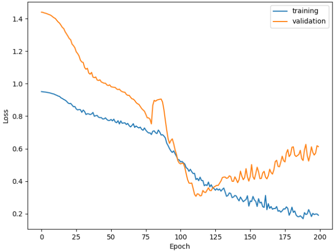
We aim to examine the performance of TL techniques

- learning the lapse features,
- learning the dynamic lapse, and
- transferring to other contract types.

The Effect of Number of Training Samples



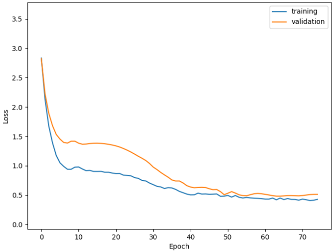
(a) 50000 samples



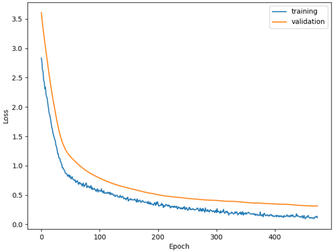
(b) 2000 samples

Figure: Learning curves of GMMB contracts with static lapse

Learning Lapse Features using Transfer Learning



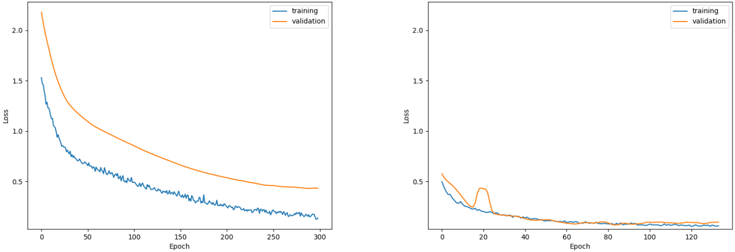
(a) Fine-tuning



(b) Layer Freezing

Figure: Learning curves of GMMB contracts with static lapse

Learning Curves: Learning Dynamic Lapse



(a) No Lapse (b) Static Lapse

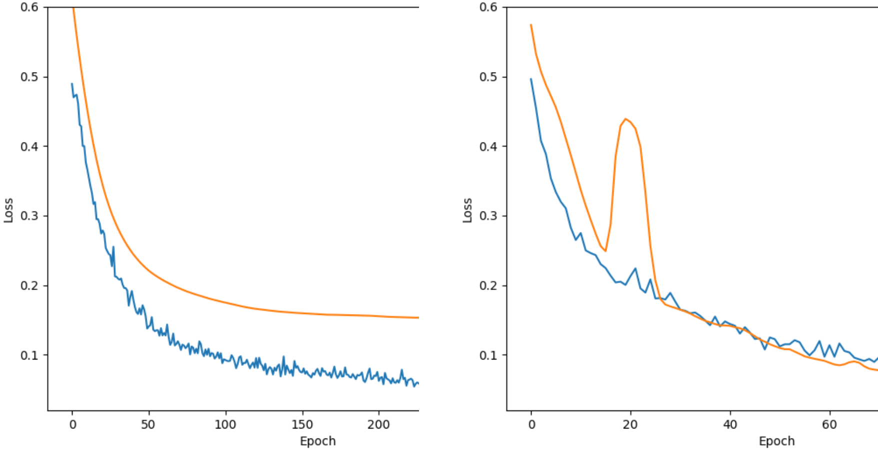
Figure: Learning curves of GMMB contracts with dynamic lapse

Results: Accuracy Comparison

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

Comparison of different TL methods on GMMB contracts (best MSE values)

Learning Curves: Effect of Similarity



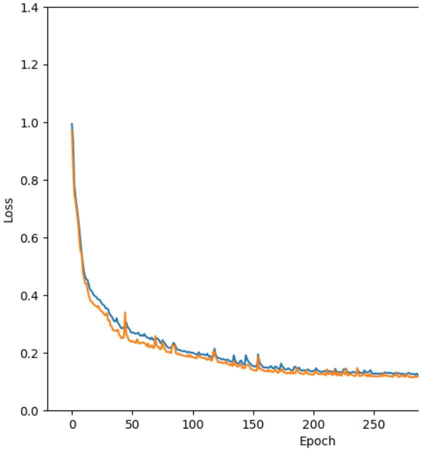
The effect of similarity between source and target on the convergence speed

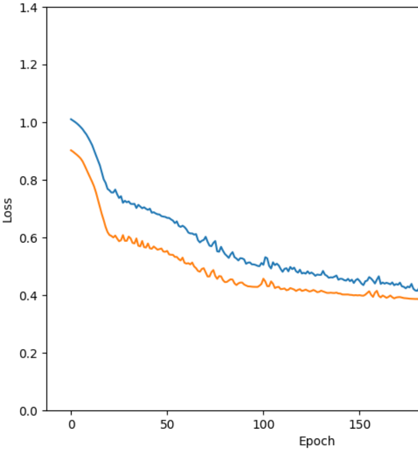
- *Left: from No Lapse; right: from Static Lapse*

Transfer Knowledge to other Contract Types

From GMMB to GMWB Contract

Learning Curves: Transfer Knowledge to other Contract Types

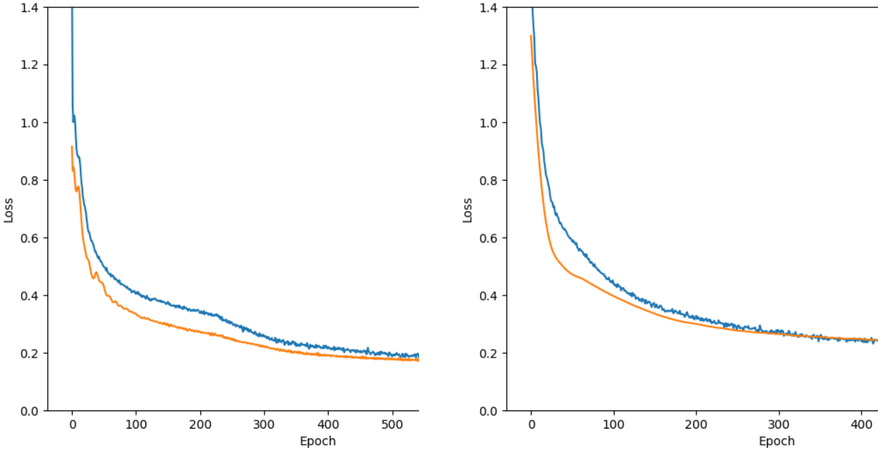




Learning curves of GMWB contracts

- *Left: 50000 samples; right: 2000 samples*

Learning Curves: Effect of Similarity



Comparison of different TL methods on GMWB contracts

- *Left: fine-tuning; right: layer freezing*

Performance Comparison (GMMB $\rightarrow$ GMWB)

Model	Training MSE	True MSE
Without TL	0.3588	0.4188
Fine Tuning	0.1690	0.1780
Layer Freezing	0.1828	0.2295
Extensive Training	0.0853	0.0726

- Fine-tuning outperforms layer freezing for dissimilar tasks
- Both TL methods better than training from scratch
- Extensive training still superior with abundant data

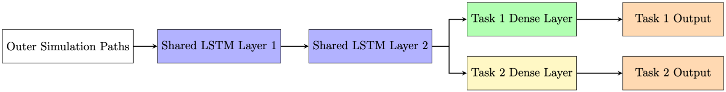
Practical Implementation

- **Framework**: TensorFlow/PyTorch implementation
- **Workflow**:
  1. Train base model on source contract
  2. Freeze early layers
  3. Fine-tune later layers on target contract
  4. Deploy for production use

PyTorch is recommended.

- It is more flexible and easier to customize
- It is more intuitive and easier to understand
- It is more efficient and easier to debug

Multi-task Learning



- LSTM layers shared across multiple tasks
- Task-specific fully connected layers
- Objective: Minimize sum of loss functions across all tasks

Multi-task Learning Framework

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

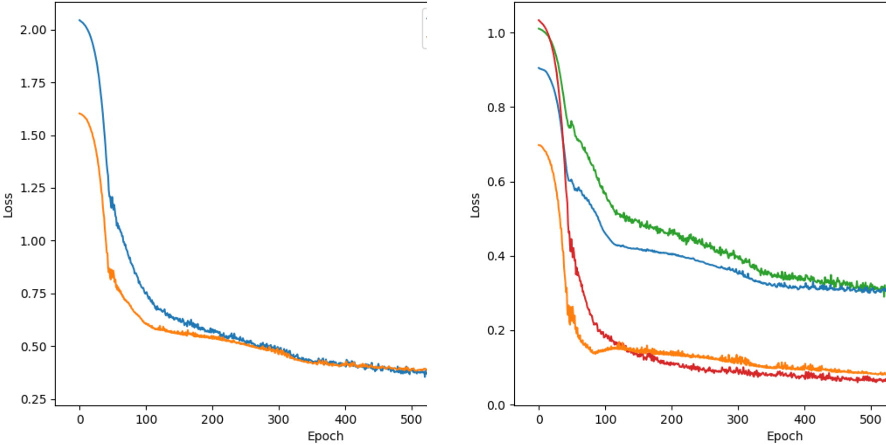
1. Train the multi-head LSTM metamodel on all K tasks simultaneously by minimizing the multi-task loss function:

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

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

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

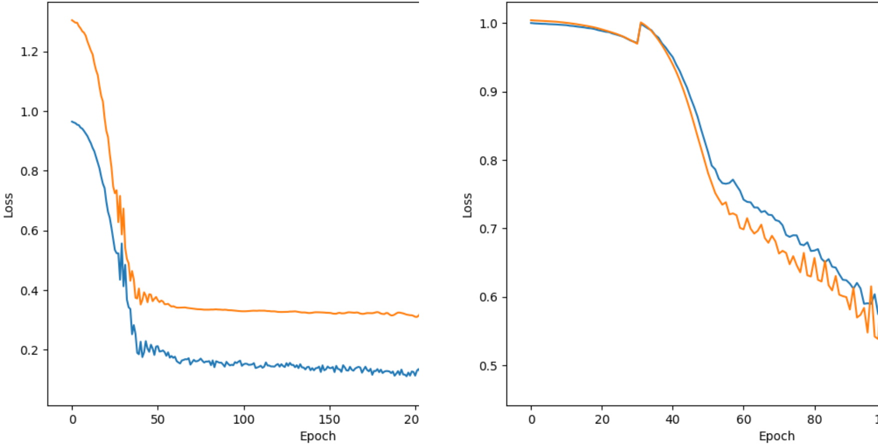
Multi-task Learning: GMMB and GMWB



Multi-task learning of GMMB and GMWB

- *Left: total MSE; right: individual MSE*

Multi-task Learning: GMMB and GMWB



Training without multi-task learning

- *Left: GMMB; right: GMWB*

Challenges & Limitations

- Determining optimal layer freezing strategy
- Handling significantly different contract structures
- Quantifying uncertainty in transfer learning predictions
- Regulatory acceptance of black-box approaches

Conclusions

- Transfer learning significantly improves metamodeling for VA contracts:
  - Faster training convergence
  - Better prediction accuracy
  - Reduced computational requirements
- Enables more frequent risk assessments and faster decision-making

Future Directions

- Incorporating domain knowledge into transfer process
- Extension to other insurance and financial products
- Multi-task learning with more than two tasks

References

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

Feng, M. and Song, E. (2020). Optimal nested simulation experiment design via likelihood ratio method. *arXiv preprint arXiv:2008.13087*.

Gordy, M. B. and Juneja, S. (2010). Nested simulation in portfolio risk measurement. *Management Science*, 56(10):1833–1848.

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

Wang, W., Wang, Y., and Zhang, X. (2022). Smooth nested simulation: bridging cubic and square root convergence rates in high dimensions. *arXiv preprint arXiv:2201.02958*.

Zhang, K., Feng, M., Liu, G., and Wang, S. (2022). Sample recycling for nested simulation with application in portfolio risk measurement. *arXiv preprint arXiv:2203.15929*.

Zhang, K., Liu, G., and Wang, S. (2021). Bootstrap-based budget allocation for nested simulation. *Operations Research*.