Xintong Li
xintong.li1@uwaterloo.ca

Dept. Statistics and Actuarial Science
University of Waterloo

UNIVERSITY OF WATERLOO

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

Supervised by Prof. Tony Wirjanto and Prof. Mingbin Feng

Thesis Defense
University of Waterloo
2025

Thank you for attending my thesis defense.
Today, I will present my work on resilient machine learning approaches for fast risk evaluation and management of financial portfolios and variable annuities.
I am supervised by Prof. Tony Wirjanto and Prof. Mingbin Feng.

Outline

My thesis focuses on improving nested simulation with supervised learning metamodels. It consists of three parts.
In the first part, I focus on comparing different nested simulation procedures for risk evaluation of financial derivatives.
In the second part, I examine deep neural network based nested simulation procedures for risk management of variable annuities.
In the third part, I propose a transfer learning based nested simulation procedure for fast adaptation to new variable annuity contracts.

Nested Simulation Procedures

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

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

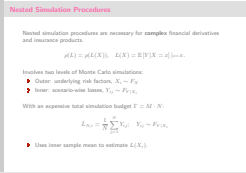
Involves two levels of Monte Carlo simulations:

- Outer: underlying risk factors, $X_i \sim F_X$
- Inner: scenario-wise losses, $Y_{ij} \sim F_{Y|X_i}$

With an expensive total simulation budget $\Gamma = M \cdot N$:

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

- Uses inner sample mean to estimate $L(X_i)$.



In the context of quantitative risk management, nested simulation is used for risk evaluation of financial derivatives and insurance products.
The nested structure is due to the loss random variable being a conditional expectation of the portfolio loss given the underlying risk factors.
In our problems, the loss L depends on the risk factors X and is the conditional expectation of the portfolio loss given the risk factors.
More specifically, X is d-dimensional and L is 1-dimensional.
Estimating a risk measure, $\rho(L)$, involves two levels of Monte Carlo simulations: a outer level generates underlying risk factors, and a inner level generates scenario-wise samples of portfolio losses.
The standard nested simulation procedure uses the inner sample mean under that scenarioto estimate $L(X_i)$.
The nested structure makes the estimation computationally expensive.
In this thesis, we focus on examining and improving the efficiency of nested simulation procedures mostly by pooling with supervised learning metamodels.

Metamodeling Approach

We focus on procedures that **pool** with **supervised learning metamodels**.

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

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

- Use metamodel predictions to estimate risk measures

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

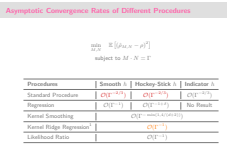
Pooling can happen in different ways, and the key idea is to use the information from different outer scenarios to improve the efficiency of nested simulation.
A supervised learning based nested simulation procedure pools with these steps.
First, the user runs the standard nested simulation procedure to generate a set of outer scenarios and inner replications.
Then, the outer scenarios and inner sample means can be treated as features and labels for metamodel training.
Finally, instead of the inner sample mean from the standard procedure, the metamodel predictions estimate the portfolio loss and the risk measures.
Most existing works focus on the convergence rate of their estimators in terms of the simulation budget.

However, we find that there is a cost of pooling that comes from metamodel training, validation, and prediction.

Asymptotic Convergence Rates of Different Procedures

$$\min_{M,N} \quad \mathbb{E}\left[(\hat{\rho}_{M,N} - \rho)^2\right]$$
$$\text{subject to } M \cdot N = \Gamma$$

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[1]	$\mathcal{O}(\Gamma^{-1})$		
Likelihood Ratio	$\mathcal{O}(\Gamma^{-1})$		



We start by comparing the asymptotic rates of different nested simulation procedures.
These are the convergence rates of the MSE of the risk measure estimator when the total simulation budget Γ goes to infinity.
The standard procedure has the same convergence rate of $\mathcal{O}(\Gamma^{-2/3})$ for all types of h considered. The rates in red are not available in the original works but are derived in this thesis.
The regression-based procedure has $\mathcal{O}(\Gamma^{-1})$ convergence rate for smooth and hockey-stick h.
It is achieved by only allowing M to grow. In their experiments, N is set to equal a constant.
The kernel smoothing procedure is the only procedure that depends on the asset dimension d.
When risk factor X is high-dimensional, the kernel smoothing procedure may converge slower than the standard procedure.
The kernel ridge regression procedure and likelihood ratio procedure have the same convergence rate as the regression-based procedure.
However, in our numerical experiments, we find that their additional cost of pooling may offset their benefit of fast asymptotic rates.
Additionally for KRR, the authors analyze convergence of absolute error in probabilistic order instead of MSE.

Key Theoretical Results

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

- Convergence in MSE:

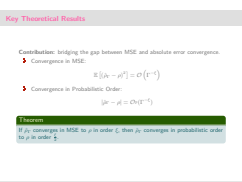
$$\mathbb{E}\left[(\hat{\rho}_\Gamma - \rho)^2\right] = \mathcal{O}\left(\Gamma^{-\xi}\right)$$

- Convergence in Probabilistic Order:

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

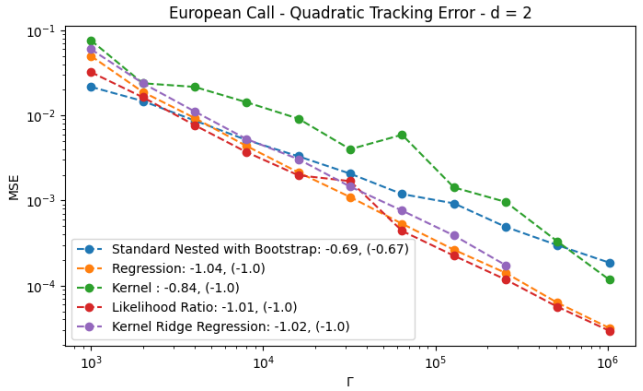
Theorem

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

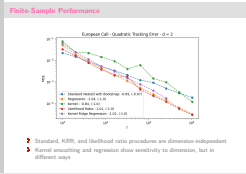


In most theoretical works, the focus is on the convergence rate of the MSE of the risk measure estimator.
However, for KRR, the authors analyze convergence of absolute error in probabilistic order.
One of the contributions of this thesis is to bridge the gap between MSE and absolute error convergence.
We first define the convergence in MSE and the convergence in probabilistic order.
And we show that the convergence rate of $\Gamma^{-\xi}$ in MSE of the risk measure estimator implies the convergence rate of $\Gamma^{-\frac{\xi}{2}}$ in probabilistic order.
We also show that the converse is not necessarily true.

Finite-Sample Performance



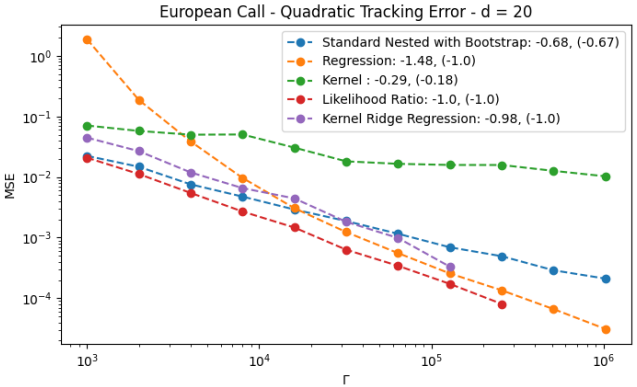
- Standard, KRR, and likelihood ratio procedures are dimension-independent
- Kernel smoothing and regression show sensitivity to dimension, but in different ways



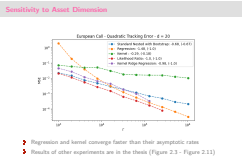
Previously, we worked on the asymptotic convergence rates of different nested simulation procedures.
In practice, the total simulation budget is finite.When the budget is finite, the theoretical convergence rate may not be the only thing we should consider.
This figure shows the performance of all 5 procedures in their estimation of the quadratic tracking error for the portfolio of European call options.It is a log-log plot of the MSE of the estimator versus the total simulation budget Γ.For each procedure, we run 1000 macro replications of Monte Carlo simulations with different total simulation budgets.
For each data point on the plot, 1000 independent replications of the procedure are used to estimate the MSE.
The slopes of the lines estimates the empirical convergence rates of the procedures.
In the legend records the slopes of the lines and the corresponding asymptotic convergence rates of the procedures.
In the most basic case, most procedures have their empirical convergence rates match their asymptotic convergence rates.

However, as we will see after, this is not the case for all experiments.

Sensitivity to Asset Dimension



- Regression and kernel converge faster than their asymptotic rates
- Results of other experiments are in the thesis (Figure 2.3 - Figure 2.11)



This figure shows the same experiment as the previous figure, but we further increase the asset dimension to 20.
The standard procedure is not affected by the asset dimension.
The kernel smoothing procedure actually converges faster than what its asymptotic convergence rate suggests.
The regression-based procedure is the only procedure that actually converges faster when the asset dimension is high. We will do a deeper dive into it in the following experiment.
It is also worth noting that the likelihood ratio and KRR procedures are dimension-independent.
While it is not shown in their empirical rate, due to their additional cost of pooling inner replications, we are not able to run these procedures when Γ is large.
In the thesis, we provide a detailed analysis of the sensitivity for all 5 procedures with more than 500 experiments.

Additional Computational Costs

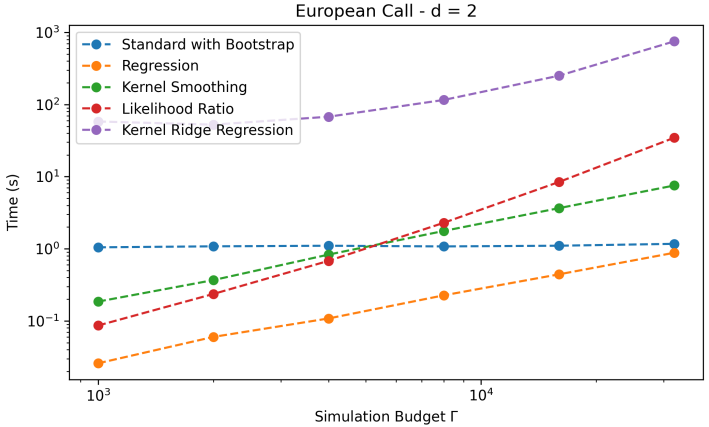
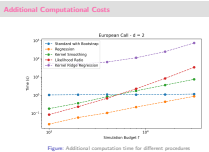


Figure: Additional computation time for different procedures



This figure shows the additional computation time for different procedures.
It includes the time for training, validation, and everything else.
The total computation time of the standard procedure does not increase fast with Γ.
It is because the computation time of estimating the optimal M and N does not increase for larger Γ.
Regression is the most efficient among metamodel-based procedures.
The likelihood ratio and KRR are more expensive due to their additional costs.
The likelihood ratio weight calculations increase quadratically with Γ.
For KRR, the cost of cross-validation is too high.

In the thesis includes a detailed analysis of the computational complexity of different procedures in terms of Γ.

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

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

In our numerical experiemenct, we have shown that the regression-based procedure is the most cost-effective among all the procedures.
It is robust and efficient to implement for limited budget sizes, and has fast empirical convergence for option portfolios.
However, the regression-based procedure is not suitable for extremely high-dimensional risk factors or complex payoffs.
The first reason is that it is hard to find a good regression basis for high-dimensional risk factors. It will involve a lot of feature engineering and expert knowledge.
The second reason is that the regression-based procedure is not able to capture the nonlinearity in the payoff.
In the next project, we will examine the performance of metamodel-based simulation procedures for variable annuities.
This is where a neural network-based procedure can be more suitable as a form of automated feature engineering.

Nested Simulation for Risk Management of VAs

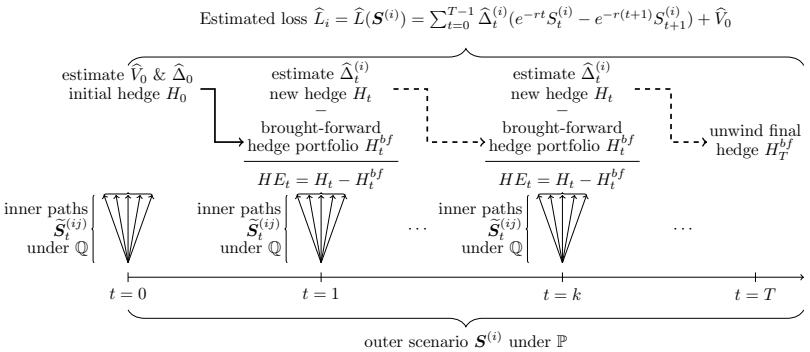
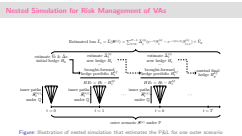


Figure: Illustration of nested simulation that estimates the P&L for one outer scenario



The main difference of the standard nested simulation of VAs can be explained by this illustration.
This is how the P&L is estimated for one outer scenario.
It starts at $t = 0$ with a simulation to estimate the initial value of the VA contract and the initial hedge. Here we are using delta-hedging.
This hedge gets carried forward and updated at each time period with another simulation.
The P&L is then estimated by adding up the hedging error at each time period to the initial value of the VA contract.
All the simulation mentioned is a single set of inner simulations for one outer scenario.
It is very expensive to run this standard nested simulation for VAs.

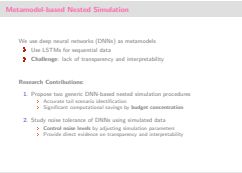
Metamodel-based Nested Simulation

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



After numerical experiments, we find that the deep neural network metamodels are suitable for this problem.
We use LSTM for the sequential data. It provides accurate tail scenario identification and significant computational savings by budget concentration.
However, it is not transparent and interpretable. Many researchers are working on this topic.
This is where we can contribute to the research.
By using nested simulation as a data generating process, we can study the noise tolerance of DNNs.
The noise level in the training labels can be controlled by adjusting the simulation parameters.
By examining the performance of DNNs with different noise levels, we can provide direct evidence on the transparency and interpretability of DNNs.

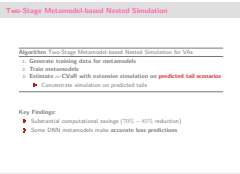
Two-Stage Metamodel-based Nested Simulation

Algorithm Two-Stage Metamodel-based Nested Simulation for VAs

1: **Generate training data for metamodels**
2: **Train metamodels**
3: **Estimate α-CVaR with extensive simulation on predicted tail scenarios**
- Concentrate simulation on predicted tails

Key Findings:

- Substantial computational savings ($70\% - 85\%$ reduction)
- Some DNN metamodels make **accurate loss predictions**



The two-stage procedure is a two-step process that is based on the idea of budget concentration.
First, we generate training data for the metamodel.
We use a fraction of the simulation budget to run the standard nested simulation procedure with M outer scenarios and N' inner replications.
We then train a metamodel using the feature-label pairs.
We use the trained metamodel to make predictions for all outer scenarios.
We then identify the predicted tail scenario set that contains the m largest predicted losses.
Finally, we run the standard nested simulation procedure on the predicted tail scenarios.

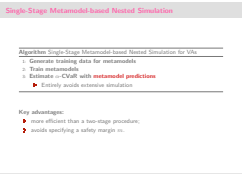
Single-Stage Metamodel-based Nested Simulation

Algorithm Single-Stage Metamodel-based Nested Simulation for VAs

1: **Generate training data for metamodels**
2: **Train metamodels**
3: **Estimate α-CVaR with metamodel predictions**
   - Entirely avoids extensive simulation

Key advantages:

- more efficient than a two-stage procedure;
- avoids specifying a safety margin m.



The single-stage procedure shares the same idea of running the standard nested simulation procedure to generate training data for the metamodel.
After a metamodel is trained, we use it to make predictions for VA contract losses accross all outer scenarios.
The risk measure is estimated using the predicted losses of the metamodel.
It is more efficient than the two-stage procedure because it avoids the second stage of the two-stage procedure.
It also avoids specifying a safety margin m, which is a feature of the two-stage procedure that we will discuss later.

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

In our experiment:

- 95%-CVaR of loss for a GMWB with a 240-month maturity
- 240-dimensional feature vector and 1-dimensional loss

We consider the following metamodel architectures:
Multiple Linear Regression, Quadratic Polynomial Regression, Feedforward Neural Network, Recurrent Neural Network, and Long Short-Term Memory Network.
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.
The deep neural network metamodels are more flexible and expressive, but they are said to be more prone to overfitting.
We will see whether this is indeed the case in our experiments.

Metamodel Performance

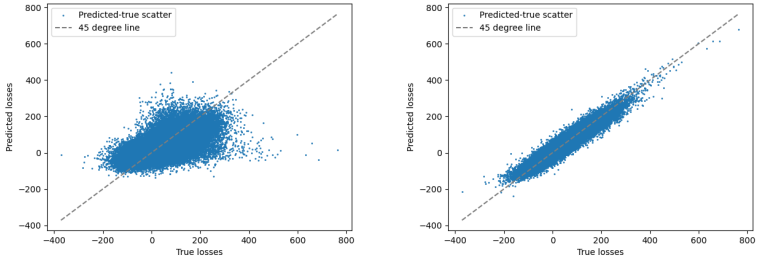
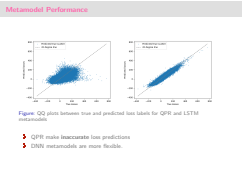


Figure: QQ plots between true and predicted loss labels for QPR and LSTM metamodels

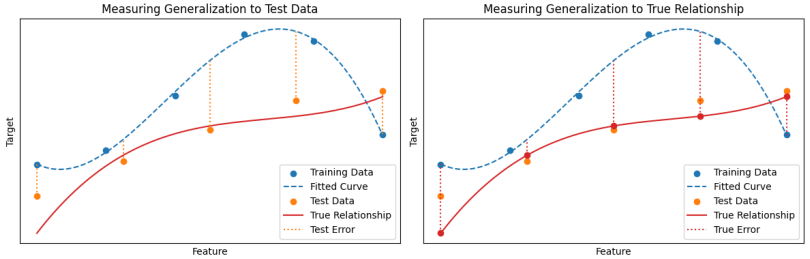
- QPR make **inaccurate** loss predictions
- DNN metamodels are more flexible.



We first take a look at their performance with QQ plots.
On the left is the QQ plot between the true loss labels and the predicted loss labels for QPR, and on the right is the QQ plot for LSTM.
Feature engineering for QPR is hardly feasible for our 240-dimensional X. Its poor performance is expected.
DNN metamodels are more flexible, and they can make more accurate loss predictions.

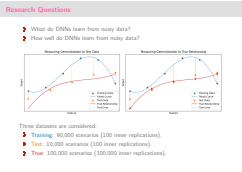
Research Questions

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



Three datasets are considered:

- Training: 90,000 scenarios (100 inner replications).
- Test: 10,000 scenarios (100 inner replications).
- True: 100,000 scenarios (100,000 inner replications).



To further clarify our research questions, we consider the following scenario:
We have a noisy training data and a noisy test data.
We want to know what the metamodel learns from the noisy data.
We also want to know how well the metamodel can make accurate loss predictions for given scenarios.
In a simulation experiment, we can control the noise level in the training data and the test data.
We also have access to the true feature-label relationship when we increase the number of replications.

We can use this true feature-label relationship to evaluate the performance of deep neural networks, which is usually not possible in practice.

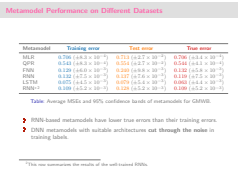
Metamodel Performance on Different Datasets

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*[2]	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.

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

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



This table summarizes the results of our experiments.
It is the most interesting finding that DNN metamodels with suitable architectures can **cut through the noise** in training labels.
The average MSEs of metamodels on the training, test, and true data are reported in the three columns, respectively.
Note that the training and test labels are noisy. The simulation noise is what makes them noisy, which is less present in the true data.
First, we note that DNN metamodels has lower training errors than the traditional regression metamodels.
This is expected because DNNs are more flexible and expressive than traditional regression metamodels.
What is more interesting is the comparison between the test and true errors.
We find that the true errors of RNN-based metamodels are lower than their test errors.
This suggests that the RNN and LSTM metamodels can learn the true feature-label relationship.
When we take another look at the table, we find that for LSTM, its true error is even lower than its training error.
This is another evidence that LSTM metamodels are able to cut through the noise in training labels and fit to the true relationship.
RNN metamodels suffer from the vanishing gradient problem. This is why we make separate rows for well-trained RNNs and poorly-trained RNNs.

Safety Margin

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

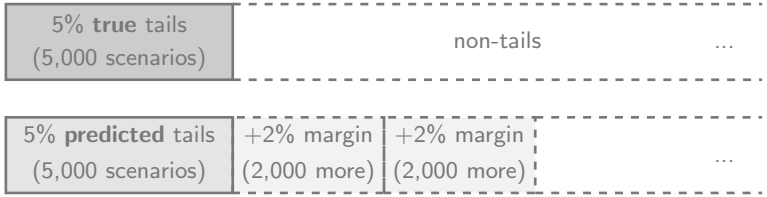
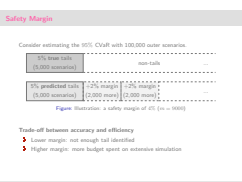


Figure: Illustration: a safety margin of 4% ($m = 9000$)

Trade-off between accuracy and efficiency

- Lower margin: not enough tail identified
- Higher margin: more budget spent on extensive simulation



For a two-stage procedure, we need to choose a safety margin m to identify the tail scenarios.
Consider estimating the 95% CVaR with 100,000 outer scenarios.
On the top is the true tail and non-tail scenarios, and on the bottom is the predicted tail and non-tail scenarios.
The 5% predicted tail scenarios are not guaranteed to contain all the true tail scenarios.
Therefore, we need to add a safety margin m to the predicted tail scenarios.
On the bottom, we add a 2% safety margin to the predicted tail scenarios.
This adds 2,000 more scenarios to the inner simulations.
We may add another 2% safety margin to be more confident.
This introduce a safety margin of 4%. 9000 predicted tail scenarios are used for extensive inner simulations in stage 2.
This is a trade-off between accuracy and efficiency.
For a good metamodel, we only need a small safety margin to be accurate in tail identification.
However, for a bad metamodel, the safety margin needs to be large to identify all the tail scenarios.
This comes at the cost of more inner simulations, which is computationally expensive.
So, one of the limitations of the two-stage procedure is the choice of safety margin.
It can be partially mitigated by using a good metamodel.

Metamodel Performance

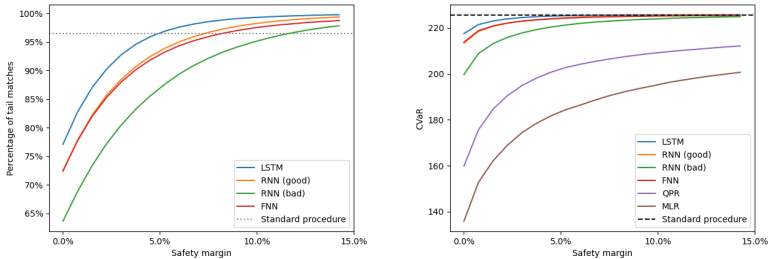
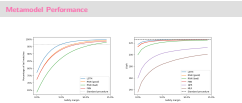


Figure: Tail Matches and CVaR Predictions for DNN Metamodels

- 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.



We now consider the tail scenario identification and the CVaR estimation for the two-stage procedure.
On the left is the tail scenario identification plot, and on the right is the plot of CVaR.
We compare the two-stage procedure with the standard procedure.
When colored lines cross the grey dashed line, it means that the metamodel is able to identify the tail scenarios as well as the standard procedure.
And it should be able to estimate the CVaR as accurately as the standard procedure.
We find that the LSTM metamodel is able to surpass the standard procedure on tail identification with a less than 5% safety margin.
However, the traditional regression metamodel is unable to identify the tail scenarios even with a high safety margin.

A well-trained RNN is better than a FNN, but the poorly-trained RNN is far worse.The CVaR estimation plot shows similar results.

Sensitivity Testing for DNNs

Simulation controls **noise level in training labels** and **number of training samples**.

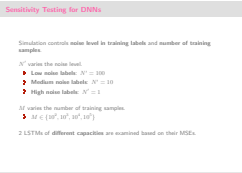
N' varies the noise level.

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

M varies the number of training samples.

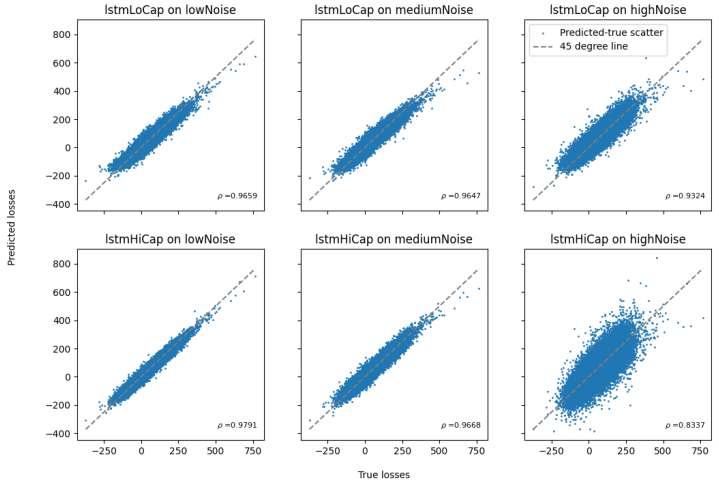
- $M \in \{10^2, 10^3, 10^4, 10^5\}$

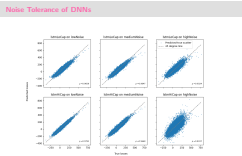
2 LSTMs of **different capacities** are examined based on their MSEs.



We now dig deeper into the research questions asked: how well do deep learning metamodels learn from noisy data?
We study this phenomenon by considering the sensitivity testing for the DNN metamodels.
The sensitivity of the DNN metamodels to the noise level in training labels and the number of training samples.
In a simulation experiment, we can vary them to see how the DNN metamodels perform.
We control the noise level in training labels by varying the number of inner replications N'.
We control the number of training samples by varying the number of outer scenarios M.
We consider two LSTM metamodels of different capacities because we want to see if the capacity of the metamodel matters for overfitting.
The high-capacity LSTM metamodel has 10 times more parameters than the regular LSTM metamodel

Noise Tolerance of DNNs





This figure shows the QQ plots of the two LSTM metamodels.
We plot the predicted losses versus the true losses for both LSTMs and different noise levels.
The high-capacity LSTM is better at low and medium noise levels, while the regular LSTM is better at high noise levels.

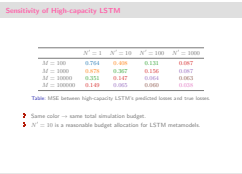
We can see on the bottom right that the high-capacity LSTM has very poor performance.

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 → same total simulation budget.
- $N' = 10$ is a reasonable budget allocation for LSTM metamodels.



This table shows the MSEs of the two LSTM metamodels on different datasets.
We modify M, the number of outer scenarios, and N', the number of inner replications.
M is the number of data points in the training set, and N' is the noise level in the training labels.
We have additional findings that the number of data points, M also matters.
And it matters more for than N' does.
In this table, the entries in the same diagonal represent the same total simulation budget.
It gives us a good indication of how much noise the LSTM metamodel can tolerate.
For a practical application, we almost want to increase M as much as possible.
However, we can see that in the diagonal of red color, we should keep N' at 10 and not lower it to 1.

We don't want the training labels to be too noisy.

Single-Stage Procedure

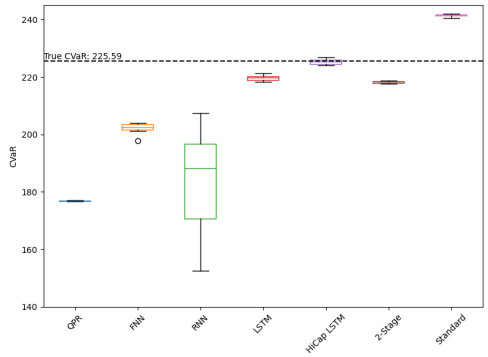
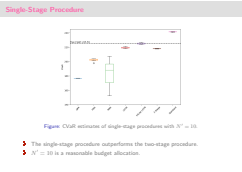


Figure: CVaR estimates of single-stage procedures with $N' = 10$.

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



This figure shows the CVaR estimates of the single-stage procedure.
We compare it with the two-stage procedure using regular LSTM metamodel with no safety margin. So we are using the 5% metamodel predicted tails to estimate the 95% CVaR with extensive simulation in stage 2.
The single-stage procedure with LSTM metamodels outperforms the two-stage procedure when noise level is moderate.
Due to its design, the single-stage procedure is more efficient than the two-stage procedure.
Setting $N' = 10$ is a reasonable budget allocation for the single-stage procedure.
Further decreasing N' to 1 deteriorates the performance of the single-stage procedure.
But it is worth noting that the 2-stage procedure can overcome this issue by increasing the safety margin.

Convergence Analysis

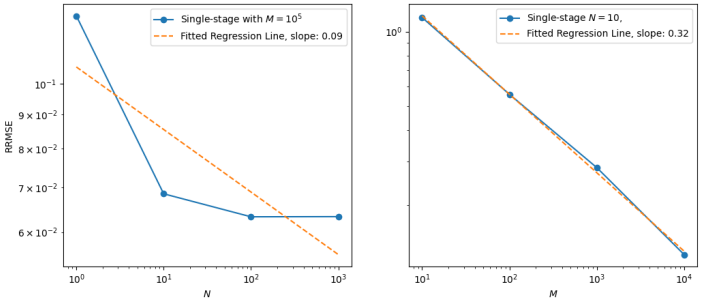
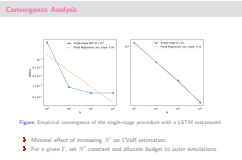


Figure: Empirical convergence of the single-stage procedure with a LSTM metamodel.

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



This figure shows the empirical convergence of the single-stage procedure with LSTM metamodels when only M or N' is varied.
On the left, we fix $M = 10^5$ and vary N'.
There is no significant difference to further increase N' when $N' \geq 10$.
On the right, we fix $N' = 10$ and vary M.
We can see that the convergence rate is around $O(M^{-2/3})$.
This finding is consistent with our previous analysis on the sensitivity of LSTM metamodels to data quality and data quantity.
Once the quality is good enough, the quantity is more important.

Conclusion

Key Findings:

- LSTMs are **resilient** to moderate levels of noise in training labels.
- DNNs can learn **true** complex dynamic hedging model.
- Two-stage procedure addresses regulatory concerns.
- Single-stage procedure is **efficient**.

Future Directions:

- Apply DNNs to other risk management tasks.
- Fast adaptation to new contracts/market conditions.

In this project, we present our extensive research into LSTM metamodels in a simulation setting.
Our findings demonstrate that LSTMs can effectively learn complex hedging strategies even with noisy training data.
The single-stage procedure offers an excellent balance of efficiency and accuracy for practical applications.
We've shown that allocating computational budget to increase outer scenarios rather than inner simulations yields better results.
These insights provide valuable guidance for implementing machine learning in financial risk management.
For the research directions, we can apply deep neural network metamodels to other actuarial applications.
We can also apply transfer learning to speed up the training process for deep neural network metamodels.

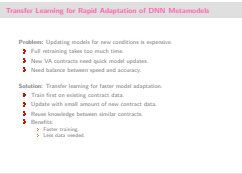
Transfer Learning for Rapid Adaptation of DNN Metamodels

Problem: Updating models for new conditions is expensive.

- Full retraining takes too much time.
- New VA contracts need quick model updates.
- Need balance between speed and accuracy.

Solution: Transfer learning for faster model adaptation.

- Train first on existing contract data.
- Update with small amount of new contract data.
- Reuse knowledge between similar contracts.
- Benefits:
  - Faster training.
  - Less data needed.



In this project, we propose a transfer learning framework for rapid adaptation of deep neural network metamodels in VA dynamic hedging.
The problem is the same as the previous project. We want to use deep neural network metamodels to approximate the complex simulation model that involves dynamic hedging.
The difference is that we want to use transfer learning to speed up the training process.
This can be beneficial for the practical applications.
Firstly, new VA contracts are continuously issued.
Secondly, market conditions are changing.
Thirdly, the computational budget is limited.
Transfer learning can help us to adapt to the new market conditions quickly and efficiently.
We can pre-train the deep neural network metamodel on a large dataset of VA contracts with abundant simulation data.
Then, we can fine-tune the metamodel on a smaller dataset of new VA contracts with limited data.
The data is often limited for new VA contracts as simulation is expensive to run.
The shared features between VA contracts can be leveraged to speed up the training process.
We can also transfer the knowledge to other contract types.
When a new VA contract is issued, we can use transfer learning to adapt the metamodel to the new contract quickly.

Transfer Learning Framework

Key Components:

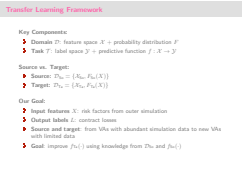
- **Domain** $\mathcal{D}$: feature space $\mathcal{X}$ + probability distribution F
- **Task** $\mathcal{T}$: label space $\mathcal{Y}$ + predictive function $f : \mathcal{X} \to \mathcal{Y}$

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 provides a formal framework for adapting models from one domain to another.
The domain consists of a feature space and a probability distribution over that space.
In our context, the feature space includes risk factors from the outer simulation.
It is equipped with a probability distribution that describes the joint distribution of the risk factors.
We don't know the exact joint distribution of the risk factors, but we can simulate outer scenarios from it using monte carlo simulation.
The task consists of a label space and a predictive function mapping features to labels.
For VA contracts, our labels are the contract losses under different scenarios.
The source domain typically has abundant data - this could be existing VA contracts with extensive simulation data.
The target domain has limited data - new VA contracts or existing contracts under new market conditions.
Our goal is to leverage knowledge from the source domain to improve prediction in the target domain.
This is particularly valuable when simulation data is expensive to generate for new contracts.
The transfer learning approach allows us to maintain accuracy while significantly reducing computational requirements.

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.

Key considerations: similarity between source and target tasks

Experiment Design:

- Source domain (50000 samples): GMMB with no lapse and GMMB with static lapse
- Target domain (2000 samples): GMMB with dynamic lapse

Fine-tuning involves using a pre-trained model as a starting point for a new task.
For VA contracts, we can fine-tune models between different contract types or market conditions.
Layer freezing is a technique where we keep some layers of the pre-trained model fixed.
Typically, early layers capture general features while later layers are more task-specific.
By freezing early layers, we preserve general knowledge while adapting task-specific layers.
In our experiment, we use the same model architecture for all tasks.
Metamodels are trained with 50,000 samples on the source domain before transferring to the target domain with only 2000 samples.
We progress from simpler to more complex models (No lapse $\rightarrow$ Static $\rightarrow$ Dynamic).
The experiments are designed to evaluate how similarity between source and target affects transfer efficiency.
This is another attempt to use simulation models as data generating processes to examine deep neural networks.

By examining the transfer learning performance, we can gain insights into which parts of the neural network are transferable, or more generally, which layers of the neural network learn which features of the VA contracts.

Results: Accuracy Comparison

Method and Setting	MSE
Extensive Training (Dynamic)	**0.0587**
Fine-tuning (Static)	0.0794
Layer Freezing (Static)	0.0763
Layer Freezing (No Lapse)	0.3361
Fine-tuning (No Lapse)	0.4894
Without TL (Dynamic)	0.2950

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

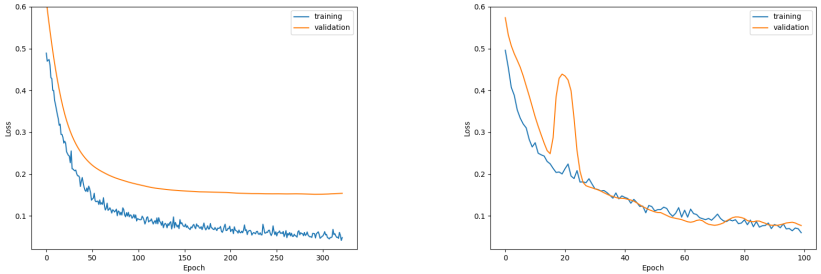
Results: Accuracy Comparison

Method and Setting	MSE
Extensive Training (Dynamic)	**0.0587**
Fine-tuning (Static)	0.0794
Layer Freezing (Static)	0.0763
Layer Freezing (No Lapse)	0.3361
Fine-tuning (No Lapse)	0.4894
Without TL (Dynamic)	0.2950

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

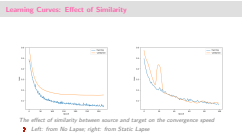
This table summarizes the MSE values for different training methods.
Extensive training with full data achieves the best performance (0.0587 MSE).
Without transfer learning, training from scratch on dynamic lapse data yields mediocre results (0.2950 MSE).
Transfer learning from static lapse models (both fine-tuning and layer freezing) performs well.
Transfer from no lapse models performs poorly, demonstrating that negative transfer can occur.
Negative transfer happens when source and target tasks are substantially different.
In such cases, knowledge from the source model can actually hinder learning on the target task.
This highlights the importance of selecting appropriate source models for transfer learning.

Learning Curves: Effect of Similarity



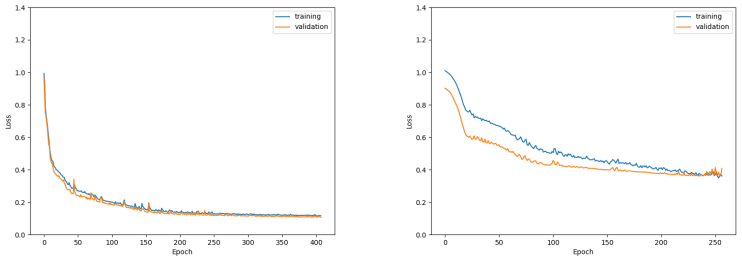
The effect of similarity between source and target on the convergence speed

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



These figures further illustrate how source-target similarity affects transfer learning performance.
The left graph shows transfer from no lapse models, which has slower convergence.
The right graph shows transfer from static lapse models, with faster convergence.
This reinforces our finding that greater similarity between source and target domains leads to more effective knowledge transfer.

Learning Curves: Transfer Knowledge to other Contract Types

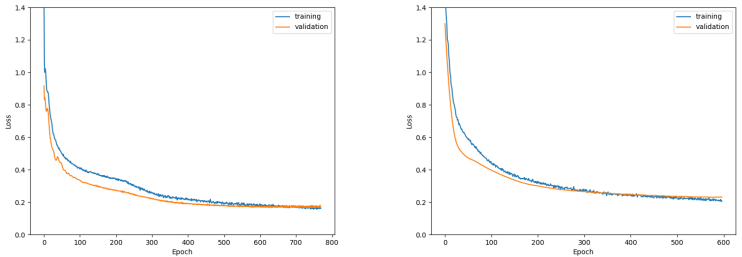


(a) 50000 samples (b) 2000 samples

Figure: Learning curves of GMWB contracts

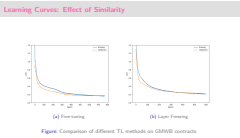
These graphs show learning curves for GMWB contracts with different training sample sizes.
Left figure (50,000 samples) shows that with abundant data, all methods eventually converge well.
Right figure (2,000 samples) demonstrates the advantage of transfer learning in data-limited scenarios.
With limited data, transfer learning methods significantly outperform training from scratch.
This is particularly relevant for new products or market conditions where historical data is scarce.

Learning Curves: Effect of Similarity



(a) Fine-tuning (b) Layer Freezing

Figure: Comparison of different TL methods on GMWB contracts



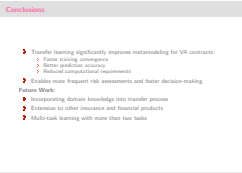
These figures compare fine-tuning vs. layer freezing approaches for GMWB contracts.
Fine-tuning (left) allows all network parameters to be updated during training.
Layer freezing (right) keeps early layers fixed and only updates later layers.
For dissimilar contracts like GMMB to GMWB, fine-tuning generally performs better.
This suggests that when transferring between different contract types, more flexibility in adaptation is beneficial.

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 Work:

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



Our research demonstrates that transfer learning significantly enhances VA contract metamodeling.
The benefits include faster convergence, better accuracy with limited data, and reduced computational needs.
These improvements enable more frequent and timely risk assessments.
This is particularly valuable in volatile markets where rapid decision-making is essential.
The approach bridges the gap between traditional actuarial methods and modern machine learning techniques.
Several promising directions exist for extending this research.
Incorporating actuarial domain knowledge could further improve transfer learning effectiveness.
The approach could be extended to other insurance products and financial derivatives.
Multi-task learning could be expanded to handle more than two contract types simultaneously.
Integration with existing risk management systems would facilitate practical adoption.
Ultimately, these techniques could transform how financial institutions manage complex product portfolios.

References

References