PRESCRIPTIVE MODELING FOR COUNTERFACTUAL INFERENCES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

By
ELIF SENA ISIK
June 2024

Prescriptive Modeling for Counterfactual Inferences
By ELIF SENA ISIK
June 2024

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Taghi Khaniyev(Advisor)

Cem Tekin

Gizem Sultan Nemutlu

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

PRESCRIPTIVE MODELING FOR COUNTERFACTUAL INFERENCES

ELIF SENA ISIK
M.S. in Industrial Engineering
Advisor: Taghi Khaniyev
June 2024

In real-life scenarios, conducting experiments or simulations to optimize outcomes can be costly in terms of time and resources. This thesis explores the utilization of trained neural networks for predictive modeling and optimization to address this challenge. The methodology involves training neural networks on historical data or simulated environments to capture complex relationships between input variables and outputs. We then employ optimization techniques to explore parameter/input spaces and identify optimal configurations for desired outputs. Importantly, this approach enables us to conduct counterfactual analyses, allowing us to assess how changes in input parameters would affect outputs. We present case studies utilizing two distinct real-life scenarios: firstly, the public simulation model FluTE, where we demonstrate the effectiveness of our approach in optimizing strategies to alleviate the spread of infectious diseases. Secondly, we tackle an assortment problem and demonstrate how decision-making processes in retail settings can be assisted by trained neural networks to maximize profitability. We then also suggest an improved methodology to control the uncertainty in predicted outputs from neural network. We utilize dropout networks to quantify variability in the output predictions and embed them into the optimization model. Computational experiments are conducted with the two case studies and customized problem specific methodologies are suggested that includes decomposition methods and heuristics.

Keywords: Mixed Integer Linear Programming, Metamodeling, Neural Networks.

ÖZET

KARSIOLGUSAL ÇIKARIM IÇIN REÇETESEL MODELLEME

ELIF SENA ISIK
Endüstri Mühendisliği, Yüksek Lisans
Tez Danışmanı: Taghi Khaniyev
Haziran 2024

Gerçek hayat senaryolarında, sonuçları optimize etmek için deneyler veya simülasyonlar yapmak zaman ve kaynak açısından maliyetli olabilir. Bu tezde bu zorluğun üstesinden gelmek için tahmine dayalı modelleme ve optimizasyon için eğitilmiş sinir ağlarının kullanımını araştırılmaktadır. Metodoloji, girdi değişkenleri ve sonuçlar arasındaki karmaşık ilişkileri yakalamak için sinir ağlarının geçmiş veriler veya simüle edilmiş ortamlar üzerinde eğitilmesini içerir. Daha sonra parametre uzaylarını keşfetmek ve istenen sonuçlar için en uygun konfigürasyonları belirlemek için optimizasyon teknikleri kullanılır. Daha da önemlisi, bu yaklaşım, girdi parametrelerindeki değişikliklerin farklı koşullar altında sonuçları nasıl etkileyeceğini değerlendirmemize olanak tanıyan karşı olgusal analizler yapmamızı sağlar. Makalede iki farklı gerçek hayat senaryosunu ele alan vaka çalışmaları sunulmaktadır: ilk olarak, bulaşıcı hastalıkların yayılmasını hafifletmek için stratejileri optimize etmede yaklaşımımızın etkinliğini gösterdiğimiz kamuya açık simülasyon modeli FluTE. İkinci olarak, bir ürün sunumu problemini ele alıyoruz ve perakende ortamlarındaki karar verme süreçlerinde karlılığı en üst düzeye çıkarmanın sinir ağları tarafından nasıl desteklenebileceğini gösteriyoruz. Daha sonra, sinir ağından tahmin edilen çıktılardaki belirsizliği kontrol etmek için geliştirilmiş bir metodoloji de öneriyoruz. Güven aralıklarını bulmak ve bunları optimizasyon modeline yerleştirmek için sönümleme ağlarını kullanıyoruz. Deneyleri yine iki örnek olay çalışmasıyla yaptık ve ayrıştırma yöntemleri ve buluşsal yöntemleri içeren özelleştirilmiş, probleme özgü metodolojiler de ortaya sunuyoruz.

Anahtar sözcükler: Karışık tam sayı doğrusal programlama, Metamodelleme, Sinir ağları.

Acknowledgement

First of all, I would like to thank my advisor Asst. Prof. Taghi Khaniyev for his support and guidance throughout my undergraduate and graduate studies, and completion of my thesis. I am glad that I had the chance to work with him, his insights and support are invaluable to me.

I would like to thank to the members of my thesis committee, Assoc. Prof. Cem Tekin and Asst. Prof. Gizem Nemutlu, for reading and reviewing my thesis.

I also want to thank TÜBİTAK for their financial support during my Master's degree studies.

This journey would have been very difficult without the constant support from my friends Elif Rana Yöner, Göksu Ece Okur, Deniz Şahin, Doğa Şahin and Şeyma Çavuşoğlu. I deeply appreciate them for always motivating me and being by my side.

I also would like to thank Gülin Yurter, Ali İlhan Haliloğlu, Doğuş Berk Koçak, Ali Eren Demir, İrem Bahtiyar, Gizem İkizler, Yunus Emre Çakır, Zeynep Göze Gürkan, Sarp Bora İlhan, Mustafa Çolak and Deniz Akkaya. I believe this period was more enjoyable thanks to the collective environment that was created by all of us. I am very lucky to have started my Master's with them.

Finally, I would like to express my deepest gratitude to my family, Fatma Işık, Kenan Işık and my sister Ayşe Zeynep Işık. I am thankful for their constant love, support and understanding during my Master's studies and while completing my thesis.

Contents

1 Introduction **1**

2 Literature Review **3**

2.1 Neural Networks 3

2.2 Monte Carlo Dropout in ANN 4

2.3 Neural Network Optimization 5

2.4 Simulation Parameter Calibration 6

2.5 Simulation Metamodeling 8

2.6 Assortment Optimization 8

3 Case Studies **10**

3.1 FluTE Parameter Calibration 10

3.2 Assortment Optimization Problem 12

4 Optimization of Point Predictions **15**

4.1 Problem Definition 15
4.2 Methodology 16
4.2.1 Neural Networks Training 16
4.2.2 Base MILP Model 18
4.3 Computational Experiments 20
4.3.1 Case Study 1: Simulation Parameter Calibration 20
4.3.2 Case Study 2: Assortment Optimization Problem 31
4.4 Discussion 33

5 Estimation Error Constrained Optimization of Point Predictions 35

5.1 Problem Definition 35
5.2 Methodology 36
5.2.1 Dropout Neural Networks 36
5.2.2 MILP Model with Dropout Networks 37
5.3 Computational Experiments 40
5.3.1 Case Study 1: FluTE Parameter Calibration 40
5.3.2 Case Study 2: Assortment Optimization Problem 45
5.4 Decomposible Structure and Extended Model 50
5.5 Discussion 56

6 Conclusion and Future Search **57**

A Calculation of Performance Metrics **63**

A.1 R^2 Calculation 63

A.2 Mean Absolute Error (MAE) Calculation 64

A.3 Mean Absolute Relative Error (MARE) Calculation 64

List of Figures

2.1 Illustration of neural network neuron 4

2.2 Simulation Parameter Calibration Process [1] 6

4.1 Neural Network Representations 17

4.2 Illustration of 4 calibration strategies 23

4.3 Illustration of single and multi output neural network model . . . 25

4.4 Predicted and Actual Error Boxplots for RS and OP methods . . 28

4.5 Illustration of PtS strategy with different proximity ranges 29

4.6 Actual Error distributions for 4 calibration strategies 30

4.7 Error distributions for different ranges in PtS method 31

4.8 Representation for change in predictions with different approximation models for a single instance 34

5.1 Dropout Neural Network Structure 36

5.2 Dropout values with different maximum difference limits 41

5.3 Dropout values with different maximum difference limits and dropout rates 42

5.4 Distribution of maximum dropout objectives with different number of dropout networks 43

5.5 Representation for target value selection 44

5.6 Distribution of objectives with different training methods and number of dropout networks 44

5.7 Objective changes with different variance limits and allowed assortment change 48

5.8 Objective changes with different range limits and allowed assortment change 49

5.9 Objective Changes for Different Allowed Assortment Changes with No Variance Limit (Optimization vs Heuristic) 50

5.10 Structure of the main model and reformulated model 51

5.11 Trial Decomposition Approach Structure with Initial Uncertainty Constraints 52

List of Tables

3.1 Input Parameters and Output Variables of FluTE selected 12

4.1 Comparison of performance metrics for Linear Regression, Regression Tree, and Neural Network models 24

4.2 R^2, MAE, and MARE values (in test sample) for single and multi-output neural network models 26

4.3 R^2 values for each output by sample size 26

Chapter 1

Introduction

In both everyday life and scientific studies, several questions emerge that explores the causality such as "How would the profit have changed if we decided on using another supplier", or "How much would the spreading rate of an illness change with different supply rates to different places?". To answer these questions different methods are developed to analyze the interactions with the interventions from real world and hypothetical outcomes, counterfactuals [2].

Conducting real-life experiments to investigate causal questions can often be impractical, expensive, or ethically challenging. This is where machine learning techniques come into play. Machine learning provides powerful tools to model complex systems and predict outcomes based on existing data. These techniques enable researchers to simulate interventions and analyze counterfactual scenarios without the need for costly or impossible experiments.

Artificial neural networks are among the most commonly used machine learning models thanks to their robustness, compatibility and convenience. ANN models are known to be universal approximators due to their capability of approximating any measurable function and achieving any desired degree of accuracy with one hidden layer [3]. They are also generalizable and have easy to replicate structure with linearizable framework while having ability to capture

non-linear relationships within high dimensional data, making them suitable for causal analysis.

The methodology proposed in this thesis involves training neural networks on historical data as predictive models, or simulated environments as metamodels, to learn patterns and relationships that underlie the real-life events. Utilizing the predictive capabilities of these networks, we then employ optimization techniques to explore parameter spaces and identify optimal configurations for desired outcomes. The use of optimization models provides an efficient exploration of feasible parameter space while making the proposed approach versatile and highly effective for various real-life problems by handling both integer and continuous interventions.

The objective of this work is to create a new general-purpose approach that can be used to conduct counterfactual analyses, allowing us to assess how changes in input parameters would affect outcomes under different conditions. We will present two case studies: one involving the calibration of the publicly available simulation model FluTE using ANNs as metamodels, and another utilizing the trained ANN on real-life data from a fast-moving consumer goods (FMCG) company for predictive modeling.

The thesis is organized as follows: first, we discuss relevant literature, briefly explain the case studies used, then present the proposed base method and the computational results through the two case studies which require slight modifications from the base model. We then propose an improved methodology based on the results of the base method. Lastly, we show the computational experiments with the same case studies.

Chapter 2

Literature Review

2.1 Neural Networks

The simplest type of artificial neural network (ANN) is a feed-forward network, which does not have loops in its network architecture. One input layer, one output layer, and the hidden layers that lie between them combine to form them. Every layer is made up of nodes, or neurons, and the general structure of these nodes is composed of three main components: (i) a set of weighted synapses, or connecting links; (ii) a summing junction, which adds the input signals that are weighted by the synapses of individual neurons; and (iii) an activation function, which modifies the output signal to capture non-linearities. Important components are shown in an illustration of a neuron in Figure 2.1. The linear (weighted) combination of input nodes from one layer of nodes to the next is summed with a bias term and then passed through an activation function to produce the final output [3].

From now on, artificial neural networks will be our preferred machine learning model because of its accuracy and computational efficiency. However, the general strategy outlined in the methodology can be adapted to other algorithms, such as tree-based models and linear regression.

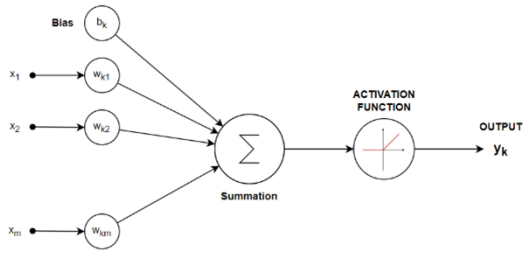

Figure 2.1: Illustration of neural network neuron

2.2 Monte Carlo Dropout in ANN

Srivastava et al. presented dropout as a method to enhance neural network generalization. The basic concept involves randomly "dropping out" (i.e., setting a unit's output to zero with a specific probability) units in a neural network during training. This lessens overfitting by preventing the units from adapting excessively. In order to maintain the expected output during testing, the outputs are scaled by p, while each unit is retained with a probability of p during training. Hence, By preventing the network from depending too much on any one neuron, it lessens overfitting and facilitates robustness [4].

To further increase dropout's efficacy, a number of variations have been suggested, such as; DropConnect [5], SpatialDropout [6], adaptive dropout methods [7], Monte Carlo dropout (MC dropout) [8].

The dropout method for estimating uncertainty in deep learning models is made by MC dropout. MC dropout was introduced by Gal and Ghahramani (2016) as a neural network approximation for Bayesian inference. Multiple forward passes and the application of dropout at test time produce a distribution of outputs from MC dropout provides an approach to estimate uncertainty in neural networks [8]. It also has been effectively used in a number of fields; including natural language processing, autonomous driving, and medical image analysis. In order to increase the robustness and reliability of deep learning models for semantic segmentation and object detection, Kendall and Gal (2017) employed MC dropout to estimate uncertainty in the models. Moreover, MC dropout has

proven useful in active learning, in which informative samples for labeling are selected based on uncertainty estimates [9].

2.3 Neural Network Optimization

In recent years, there has been many studies utilizing the integration of neural networks (NNs) with optimization models, specifically mixed-integer linear programming (MILP) [10, 11, 12, 13]. This hybrid approach solves difficult decision-making problems by combining the optimization capabilities of MILP solvers with the predictive power of NNs.

Strong mixed-integer programming formulations for trained neural networks were developed by Anderson et al. in 2020. Creating MILP formulations that faithfully capture the behavior of neural networks with ReLU activation functions was the main goal of their work. These formulations allow for more accurate and efficient optimization by giving a more accurate representation of the decision boundaries of the neural network [14]. We will also use their model as a guide for our base approach in this thesis.

Another similar study introduced a novel technique for optimizing objective functions derived from trained ReLU neural networks. ReLU networks are well-known for having a piecewise linear structure, which makes optimization strategies applicable to them. The authors estimated the objective function that the neural network represented using a sampling-based method [15].

Incorporating an approach similar to dropout networks, one paper utilizes ensemble of neural networks to efficiently control the uncertainty in neural networks within an optimization setting. By expressing the ensemble of networks as a MILP problem, the solutions are obtained over combined outputs of neural networks. This approach further improves the reliability of predictions and the decision making process[16]. A similar approach will be utilized using dropout networks in this thesis.

2.4 Simulation Parameter Calibration

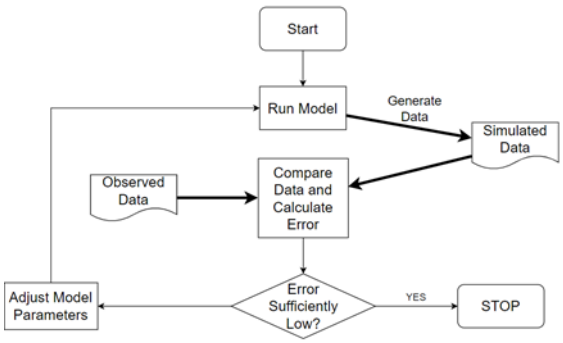

Figure 2.2: Simulation Parameter Calibration Process [1]

Figure 2.2 provides a summary of the parameter calibration procedure. An initial set of input parameters is used to initiate the process. Next, the simulation model is run using the current set of parameters at each iteration, and the output of the simulation is compared to the observed output. The calibration process is stopped if the difference in error between the two outputs is small enough; if not, the current parameter set must be changed using a predefined method. This adjustment, or calibration, is done at the "Adjust model parameters" highlighted component. There are a few different ways to do this, including automated calibration methods and manual iterative calibration [1]. In the former case, the simulation output is adjusted through trial and error until it adequately reflects the real data, based on the user's experience; in the latter case, the simulation output is adjusted through specific intrusive tests and measurements, as well as customized testing and analytical procedures [17].

Grid search, also known as parameter sweep, is among the most basic automated calibration methods. Because of its ease of use and simplicity, the medical community has embraced it widely. Using this method, all possible combinations of values for all parameters are explored by splitting the domain of each parameter into equally spaced values (i.e., the simulation model is run for each parameter combination). For instance, $10^3 = 1000$ possible parameter combinations are produced by a grid search using 1-unit increments for three input parameters, each of which takes values in the range of 1 to 10. Even for a moderate number of

parameters, it is frequently impractical to implement complete parameter sweep due to the exponential growth of parameter combinations with the number of input parameters, despite the method's simplicity [1].

Not only is it impractical to explore the entire parameter combination grid, but the parameter sweep method usually covers the entire space in a uniform manner. The need for smart allocation of limited computational resources grows with a simulation model's complexity, or run time. There are more complex approaches in the literature that focus more of their time and energy on optimizing configurations that have worked in the past, disregarding areas of the parameter space that are not likely to produce global optima.

For instance, Malleson introduces automated techniques like genetic algorithms, simulated annealing, and hill climbing. The technique known as "hill climbing" starts with a random combination of parameters and then modifies one of them slightly. Adopted if the modification lowers error and enhances model performance. A change is discarded if it results in an increase in error. Until no change to the local parameter reduces the error, this process is repeated. In contrast to hill climbing, which always selects the best option among those presented, simulated annealing and genetic algorithms permit the adoption of a neighboring solution, even if it results in an increase in error. The problem of becoming trapped at the local optima that is seen in hill climbing can be avoided by this tactic [1].

The Bayesian calibration method, which uses the Bayes theorem and observed data to derive posterior distributions for unknown parameters, is another approach used. The prediction can handle all factors of uncertainty through the use of prior input distribution, and the Bayesian calibration approach corrects model inadequacy due to discrepancies between model predictions and observed data. These two advantages make the Bayesian calibration approach more effective than the traditional method [18, 19].

2.5 Simulation Metamodeling

A simulator's metamodel, sometimes referred to as a surrogate model, is a simplified representation of the simulator. An approximation of the input/output function of the underlying simulation model can also be conceptualized as a metamodel. Reducing the cost, time, and effort associated with conducting a simulation study is the primary goal of simulation metamodeling [20]. A variety of metamodeling techniques for simulation metamodeling are covered in the literature. A comparison of the Gaussian process emulator (GPE), support vector machine (SVM), neural network (NN), multiple linear regression model (MLR), and multivariate adaptive regression splines (MARS) is presented by Lim and Zhai [18]. Polynomial regression (PR), multivariate adaptive regression splines, kriging (KR), radial basis function networks (RBF), and neural networks are the five metamodel approaches that Van Gelder et al. compare [21].

Since artificial neural networks (ANN) work with most simulation models, they are one of the most popular metamodeling techniques. Applications of ANN metamodels in various domains, including healthcare, production, and energy systems, are available in the literature [3, 22, 23].

2.6 Assortment Optimization

In the assortment optimization problem, a subset of a wide range of products is chosen to be presented to customers. Traditional approaches to solve the assortment problem relied on basic rule-based systems or heuristic methods. However, as retail environments became increasingly complex, mathematical modeling became a popular approach to solve this issue more effectively [24]. By presenting the issue as an optimization task to maximize profit, mathematical models are essential to assortment optimization. In order to maximize revenue while taking into account constraints like inventory levels and shelf space, early models mostly utilized linear programming techniques [25]. More advanced techniques

that include mixed-integer programming (MIP) and nonlinear optimization are also utilized as a result of improvement in mathematical modeling recently [26].

Machine learning (ML) also has become an important complement to mathematical modeling. Without explicit programming, ML models—in particular, supervised and reinforcement learning algorithms—offer the capacity to identify patterns and relationships from data [27]. In order to determine the best product assortments, ML models can examine past sales data, customer behavior, and market trends.

Based on past data, supervised learning algorithms, including neural networks, decision trees, and linear regression, can predict product demand. For example, characteristics such as historical sales, seasonal patterns, and price adjustments can be utilized to predict future sales of specific products [28]. With the aid of this predictive power, retailers can more precisely meet customer demand and optimize expected revenue by selecting which products have in their assortment.

In practical applications, hybrid approaches that integrate machine learning and mathematical modeling—such as optimization with deep learning-based demand forecasting—have demonstrated impressive results [29]. For instance, to determine the optimal assortment, deep learning models can be used to forecast demand for each product then these forecasts can then be used as inputs in a mathematical optimization model [30]. These hybrid models can dynamically update and adjust to new data by utilizing ML techniques, guaranteeing that the optimization process stays relevant as consumer preferences and market conditions change [31]. Furthermore, the efficiency and scalability of mathematical models can be improved by integrating ML. Large datasets can be preprocessed and their dimensionality can be reduced by ML algorithms, which makes them easier to handle for optimization algorithms.

Chapter 3

Case Studies

3.1 FluTE Parameter Calibration

Through the use of virtual environments for experimentation and behavior prediction under various scenarios, simulation models enable researchers and practitioners to obtain insights into complex systems without requiring for expensive, and often impractical real-world experiments. Ensuring the validity of simulation models, which involves comparing their output with observed real-world data, is one of the most significant challenges when using them. In order to improve the agreement between the simulated outputs and the observed/target data, validating a simulation model necessitates a proper calibration of its many parameters, many of which have a range of values. Since it requires running the underlying simulation model for numerous parameter combinations, which can take anywhere from several minutes to several days, this calibration process can often be computationally expensive. To alleviate this problem the proposed methodology can be utilized. For this thesis, we have done the computational experiments with a specific simulation model, FluTE.

Simulation Model

We experimented with a particular simulation model that is frequently employed in the epidemic studies in order to assess the effectiveness of the calibration strategies previously described: Developed by Chao et.al., FluTE is a freely accessible stochastic influenza epidemic simulation model. It mimics how the influenza virus spreads throughout the main cities in the continental United States [32]. The simulation seeks to offer helpful information regarding:

- What percentage of people are susceptible to an illness or become infected?
- To what extent can a given transmissibility rate of an epidemic be controlled by vaccination coverage?
- Which population group needs vaccinations the most to lower the rate of illness strikes overall?
- How long can one hold off on responding to an outbreak?
- What range of transmissibility rates is a given pandemic strategy capable of controlling?

The simulation model establishes smaller mixing groups, such as neighborhoods and households, with varying transmission probabilities and divides the population into communities of 500–3000 people. Additionally, it separates the population into age groups: 0–4, 5–18, 19–29, 29–64, and 64+. For every person, the model incorporates a probability of traveling of varying lengths. The afflicted individuals are separated into two cohorts based on their infectiousness: asymptomatic and symptomatic. A person recovering from influenza is expected to take six days, and those who are asymptomatic may develop symptoms at some point.

42 input parameters overall, including R0 (transmissibility rate), vaccination fraction, response delay, etc., are required for the full original model. Following that, 16 output values are given, including the overall symptomatic attack rates and the percentage of pregnant women who are symptomatic by age group. Ten million people can run a simulation in under two hours. In order to train a metamodel and generate a training sample for this study's proof-of-concept,

thousands of simulation runs were necessary. This made it necessary to use a simulation model that, for a given combination of input parameters, had a relatively short runtime. As a result, we reduced the number of participants in the original simulation model to 500,000, fixed other parameter values based on values from the literature, and included only six input parameters. We also eliminated nine output variables that exhibited notable variability in relation to the chosen parameters. A single simulation model run took about 5–6 minutes with this reduction. Table 3.1 show the list of input parameters and output variables we used. For more information on the FluTE simulation model, refer to the Chao et. al. article from 2015.

Input	Description
1	Transmissibility rate of the disease
2	Fraction of population to vaccinate
3	Number of days to wait before initiating reactive strategies
4	Number of days it takes medical personnel to ascertain a symptomatic individual
5	Fraction of working-age adults that belong to the "essential workforce and are prioritized for receiving vaccine
6	Number of vaccinations that can be administered daily by available workforce
Output	**Description**
1	Total symptomatic attack rate
2	Total symptomatic attack rate - infant
3	Total symptomatic attack rate - school age
4	Total symptomatic attack rate - adult
5	Total symptomatic attack rate - elderly
6	Symptomatic pregnant individuals - school age
7	Symptomatic pregnant individuals - adult
8	Symptomatic high risk and pregnant individuals - school age
9	Symptomatic high risk and pregnant individuals - adult

Table 3.1: Input Parameters and Output Variables of FluTE selected

3.2 Assortment Optimization Problem

For our second case study, we focused on the assortment optimization problem. As mentioned is Section 2.6, assortment optimization is an important problem

in retail and inventory management, which aims to choose the ideal combination of products to offer in a store or catalog. The objective is to maximize total profit or other business goals, like market share or customer satisfaction, while taking into account a variety of constraints which may include shelf space, budget, and product compatibility. In order to identify the optimal assortment that maximizes profitability, this problem requires evaluating customer preferences, sales information, and product attributes. We used a specific dataset to make computational experiments. The details on dataset generation will be provided in the next parts, with the problem specific constraints that are considered in the optimization problem.

Dataset Generation

For this case study we generated artificial data based on an extensive dataset from an FMCG (fast-moving consumer goods) company. The dataset consists of multiple essential elements that enable a comprehensive examination of product diversity, demographic factors, and sales outcomes. It consists of 112 binary variables that specifically indicate whether or not a given product was included in the assortment over time. The dataset also includes 46 continuous variables that represent different neighborhood demographics, including population density, average income, and distance to schools. In addition, it offers a matrix of time series data that records sales for each of the 112 products over a 12-month period. As the output, a gross sales value (GSV) is provided for each month.

Initially, we conducted an analysis to understand the importance of each input variable. This involved training two neural networks based on the input groups. The first neural network, an LSTM (Long Short-Term Memory) network, was trained using the time series sales data. The second network, a feed-forward neural network, was trained using the 46 continuous demographic variables. Then, the binary vector for assortments was also incorporated into another neural network that received the combined outputs of the first two networks. The gross sales value was predicted as an output by this final network for every month.

We tried various network configurations in order to assess the prediction accuracies. In the end, we decided to utilize just the assortment data. The choice was made considering that, although adding more input variables greatly increased the size and complexity of the optimization model in our methodology, it did not significantly increase prediction accuracy. We were able to maintain a balance between computational efficiency and accuracy by simplifying the dataset. After we decided to use only the assortment data, we generated artificial binary assortment vectors using the information on the given dataset. In the end, we generated a dataset with 112 binary input variables representing the assortment information of different products and one output for gross sales value. Computational results given in this thesis were done by using this dataset, however, the original dataset can also be easily incorporated to the suggested methodology.

Chapter 4

Optimization of Point Predictions

4.1 Problem Definition

Lets assume a real life system $z = R(x)$, with $x = (x_1, x_2, \ldots, x_N)$ as the vector of interventions (inputs) and $\mathbf{z}$ as the system's output vector. For any problem involving finding input combinations to find hypothetical system outputs, the optimization model can be formulated as follows (assume minimization problem);

$$x^* = \arg\min_{x \in \mathcal{X}} R(x)$$

where the feasible region of the input interventions is represented by $\mathcal{X}$. For majority of real-life systems mathematical expression of the system R is not possible due to high complexity between the interventions and outcomes. To solve this problem, approximation of the real-world system that can replicate the hidden structure of them can be used. Machine learning models are commonly used to approximate the relationship between the inputs and outputs of a system. Let $L(x)$ be a machine learning model that efficiently approximates $R(x)$ globally, i.e.,

$$L(x) \approx R(x), \forall x \in X$$

Let $\tilde{x}^*$ represent the optimal parameter combination derived from the problem:

$$\tilde{x}^* = \arg\min_{x \in X} L(x)$$

Then, $\tilde{x}^*$ should reasonably approximate x^*; thus, $\tilde{x}^* \approx x^*$.

4.2 Methodology

4.2.1 Neural Networks Training

This study utilizes feed-forward neural network as the main machine learning model. Hence, in this section only the ANN models' detailed structure will be explained. *Training* is the process of determining the network weights w_{st}^k and the bias terms b_t^k such that, for a set of training observations, the total error (calculated as a function of the difference between the actual and the predicted outcomes) is minimized. The network coefficients are used to approximate the system outputs given the input parameters. The structure of feed-forward neural networks can be expressed mathematically as follows.

Let us look at an example feed-forward neural network, where the hidden layer 1 (2) has L_1 (L_2) internal nodes, the input layer has N input nodes, the output layer has P output nodes, and so on. The representation can be seen in Figure 4.1. Let w_{st}^k denote the set of weights between two successive layers that are used to compute the weighted sum of the inputs. The final value of a node is obtained by adding this weighted sum to a bias term b_t^k and passing it through an activation function. For instance, the following formula is used to determine the output of the first node in hidden layer 2:

$$y_1^2 = \sigma(b_1^2 + w_{11}^1 y_1^1 + w_{21}^1 y_2^1 + \cdots + w_{L_1 1}^1 y_{L_1}^1)$$

where σ is the activation function that presents nonlinearity in neural networks. The rectified linear unit (ReLU) function is the activation function commonly

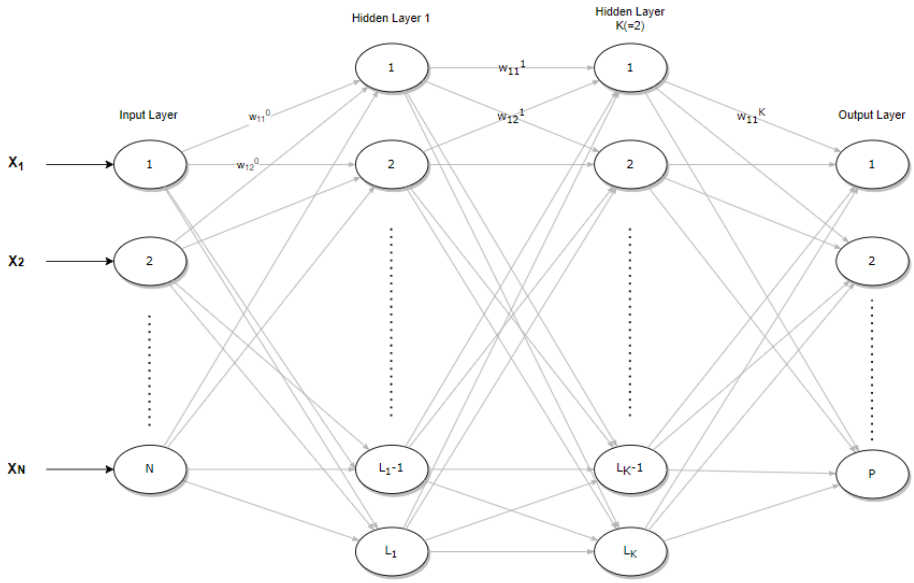

Figure 4.1: Neural Network Representations

used in feed-forward neural networks. It can be expressed explicitly as follows:

$$\sigma(x) = \max(0, x), \quad \forall x \in (-\infty, +\infty)$$

ReLU will also be used in this study as the hidden nodes' activation function. Assume that we have a trained neural network with K hidden layers (each layer k having L_k nodes) and the network coefficients w_{st}^k and b_t^k. For a given input vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, the hidden nodes' values can be computed as follows:

$$y_t^1 = \max(0, b_t^1 + \sum_{i=1}^{N} w_{it}^0 x_i), \quad \forall t = 1, 2, \ldots, L_1$$

$$y_t^{(k+1)} = \max(0, b_t^{(k+1)} + \sum_{s=1}^{L_k} w_{st}^k y_s^k), \quad \forall k = 1, 2, \ldots, (K-1), \quad \forall t = 1, 2, \ldots, L_k$$

We only use the linear activation function for the output layer because, depending on the specific application, the output may take on negative values. As a result, the P output node values are computed or predicted by the trained neural

network as follows:

$$z_j = b_j^{(K+1)} + \sum_{s=1}^{L_K} w_{sj}^K y_s^K, \quad \forall j = 1, 2, \ldots, P$$

4.2.2 Base MILP Model

After training the neural network model, the coefficients are used to make predictions inside the MILP model. The parameters and decision variables are defined as in following tables. For the sake of generalization, let x be the set of all decision variables, $\mathcal{X}$ be the feasible region and $\mathcal{F}$ be the representation for objective function in a specific problem.

Parameters		
M	An arbitrarily large number, e.g. $M = 10^6$	
N	Number of input parameters	
P	Number of outputs	
K	Number of hidden layers	
L_k	Number of nodes in the (hidden) layer k $L_0 = N$ and $L_{(K+1)} = P$	$\forall k = 0, ..., (K+1)$
w_{st}^k	Weight between node s in layer k and node t in layer (k+1)	$\forall k = 0, ..., (K+1)$,$\forall s = 1, ..., L_k$, $\forall t = 1, ..., L_{(k+1)}$
b_t^k	Bias value for node $t = 1, ..., L_k$ in layer k	$\forall k = 1, ..., L_k$
lb_i	Lower bound value for i^{th} input parameter	$\forall i = 1, ..., N$
ub_i	Upper bound value for i^{th} input parameter	$\forall i = 1, ..., N$

Decision Variables		
x_i	Value of i^{th} input parameter	$\forall i = 1, ..., N$
y_t^k	Value of node t in hidden layer k	$\forall k = 0, ..., (K+1)$
u_t^k	Binary variable indicating if node t in layer k has a nonzero value	$\forall k = 0, ..., K$, $\forall t = 1, ..., L_k$
z_j	Value of j^{th} output	$\forall j = 1, ..., P$

The mathematical optimization model is written inspired by Anderson et al. (2020). The values of the hidden nodes are represented by the decision variables y_t^k, which are computed using x_i, the network coefficients w_{st}^k, and the bias terms b_t^k. The binary variables u_t^k represent the state of activation function of a hidden node, i.e., whether or not $y_t^k > 0$. The neural network's predicted outputs are represented by the variables z_j. The objective function (1) in model is given in generalized format and can be adapted to specific problems. Constraints (2)

through (7) ensure that the network coefficients and node values from the preceding layer are correctly used to calculate the hidden node values. The non-linearity in the neural network caused by the ReLU function is effectively linearized by these constraints. The model incorporates a binary variable called u_t^k for every hidden node in order to appropriately account for this non-linearity. Constraints (8) use the node values from the last hidden layer and the associated network coefficients to predict the output node values. Constraints (9)-(12) provides the domains of base decision variables. Finally, constraint (13) is given to represent any problem specific constraint set.

$$\min / \max \quad \mathcal{F}(x) \tag{1}$$

$$\text{s.t.} \quad y_t^1 \geq b_t^1 + \sum_{i=1}^{N} w_{it}^0 x_i \qquad \forall t = 1, ..., L_1 \tag{2}$$

$$y_t^1 \geq b_t^1 + \sum_{i=1}^{N} w_{it}^0 x_i + M(1 - u_t^1) \qquad \forall t = 1, ..., L_1 \tag{3}$$

$$y_t^1 \leq M u_t^1 \qquad \forall t = 1, ..., L_1 \tag{4}$$

$$y_t^{(k+1)} \geq b_t^{(k+1)} + \sum_{s=1}^{L_k} w_{st}^k y_s^k \qquad \forall t = 1, ..., L_k, \quad \forall k = 1, ..., K-1 \tag{5}$$

$$y_t^{(k+1)} \geq b_t^{(k+1)} + \sum_{s=1}^{L_k} w_{st}^k y_s^k + M(1 - u_t^{(k+1)}) \qquad \forall t = 1, ..., L_k, \quad \forall k = 1, ..., K-1 \tag{6}$$

$$y_t^{(k+1)} \leq M u_t^k \qquad \forall t = 1, ..., L_k, \quad \forall k = 1, ..., K \tag{7}$$

$$z_j = b_j^{(K+1)} + \sum_{s=1}^{L_K} w_{sj}^K y_s^K \qquad \forall j = 1, ..., P \tag{8}$$

$$lb_i \leq x_i \leq ub_i \qquad \forall i = 1, ..., N \tag{9}$$

$$y_t^k \geq 0 \qquad \forall k = 1, ..., K, \quad \forall t = 1, ..., L_k \tag{10}$$

$$u_t^k \in \{0, 1\} \qquad \forall k = 1, ..., K, \quad \forall t = 1, ..., L_k \tag{11}$$

$$z_j \text{ unrestricted} \qquad \forall j = 1, ..., P \tag{12}$$

$$x \in \mathcal{X} \tag{13}$$

4.3 Computational Experiments

4.3.1 Case Study 1: Simulation Parameter Calibration

4.3.1.1 Problem Definition and Customization of the Methodology

We have a simulation model $z = S(x)$, with $x = (x_1, x_2, \ldots, x_N)$ as the vector of input parameters and z as the simulation's output (or a vector $\mathbf{z}$ of outputs). The calibration task we have can be formulated as an error minimization problem given a target output $\hat{z}$ observed in the real world. The error is defined as $\epsilon = \|z - \hat{z}\| = \|S(x) - \hat{z}\|$. In particular, when the simulation has a single output, determining the best parameter combination calibrated with respect to the target output $\hat{z}$ problem can be formulated as follows:

$$x^* = \arg\min_{x \in \mathbf{X}} \|S(x) - \hat{z}\|$$

where $\mathcal{X}$ represents the feasible region of the input parameters. In a similar way, if the simulation results in more than one output, say $z = (z_1, z_2, \ldots, z_P)$, then the total error can be expressed as the weighted sum of the errors for each output,

$$\epsilon = \sum_{j=1}^{P} w_j \epsilon_j = \sum_{j=1}^{P} w_j |z_j - \hat{z}_j| = \sum_{j=1}^{P} w_j |S_j(x) - \hat{z}_j|$$

where w_j represents the relative weight assigned to the output j, and $S_j(x)$ represents the j-th output of the simulation. In order to keep things simple and maintain generality, we will assume $w_j = 1, \forall j = 1, 2, \ldots, P$, in the rest of the parts of the thesis. The problem can be stated as follows:

$$x^* = \arg\min_{x \in X} \sum_{j=1}^{P} |S_j(x) - \hat{z}_j|$$

Approximation is the only option available for the majority of realistic simulation models since it is impossible to find a closed form expression for the optimal x^*. Such approximations to the original simulation models are known as metamodels. Let $M(x)$ be a metamodel that adequately approximates $S(x)$ globally, i.e.,

$$M(x) \approx S(x), \forall x \in X$$

Let $\tilde{x}^*$ denote the optimal parameter combination found from the following problem:

$$\tilde{x}^* = \arg\min_{x \in X} \|M(x) - \hat{z}\|$$

Then, $\tilde{x}^*$ should adequately approximate x^*; hence, $\tilde{x}^* \approx x^*$.

Change in the Base Model

Considering the specific problem, following parameters and decision variables are added to the model;

Additional Parameters		
$\hat{z}_j$	Target value for output j	$\forall j = 1, ..., P$

Additional Decision Variables		
δ_j	Difference between predicted and target output j	$\forall j = 1, ..., P$

The objective $\mathcal{F}$ is defined as; *minimize* $\sum_{j=1}^{P} \delta_j$

Two additional constraints are added to the model to linearize the absolute difference between the target and predicted outputs;

$$\delta_j \geq \hat{z}_j - z_j \qquad \forall j = 1, \ldots, P \tag{13.1}$$

$$\delta_j \geq z_j - \hat{z}_j \qquad \forall j = 1, \ldots, P \tag{13.2}$$

The input parameter combination $\tilde{x}^* = (\tilde{x}_1^*, \tilde{x}_2^*, \ldots, \tilde{x}_N^*)$ that has the smallest total predicted error (i.e., $\delta = \sum_{j=1}^{P} |M(\tilde{x}^*) - \hat{z}_j|$) with respect to the target outputs is obtained by solving this mathematical optimization model for a given trained neural network metamodel and the target output values. We expect $\tilde{x}^*$ to be close to x^* if the neural network metamodel sufficiently approximates the underlying simulation model, assuming the optimally calibrated input parameter combination

x^* calibrated based on the actual error (i.e., $x^* = \arg\min_{x \in \mathcal{X}} \sum_{j=1}^{P} |S_j(x) - \hat{z}_j|$) is unique.

The use of a metamodel alone for calibration could result in subpar results as we are not always in control of a metamodel's accuracy. Even in such instances, raising a metamodel's accuracy typically comes at the cost of more computing power during the training process. So, we also suggest using the input parameter combination found via optimal metamodel calibration, or x^*, as a starting point for finding better calibrated input parameters relative to the real simulation error. This essentially translates to reducing the search space that real simulation runs will investigate.

Problem Specific Hybrid Methodologies

For the computational efficiency, let's say there is a set budget of r simulation runs. How can these expensive r simulation runs be used most effectively? One common method for baseline calibration involves selecting a random point r from the whole feasible region for the input parameters, running the simulation model for those random points, and selecting the point that yields the lowest actual error concerning the desired outputs. This approach will be known as RandomlySimulate (RS). An alternative approach is the matemodel-based optimization method that was just mentioned and will be referred to as OptimallyPredict (OP) from now on. We also suggest two hybrid methods that combine the two baseline strategies:

Simulate-then-Predict (StP) is the descriptive term of the first hybrid technique (the words "optimally" and "randomly" are omitted for conciseness). As the name implies, after executing the RS strategy (i.e., using our budget of r simulation runs in this stage), we choose the point with the smallest actual error as an anchor. We then execute the OP strategy close to the anchor point (by imposing lower and upper bounds on the input parameters). Lastly, we select the calibrated input parameter combination as the point that OP finds.

Predict-then-Simulate (PtS), the second hybrid approach, combines the two base strategies in the opposite order. To be more precise, we apply OP to the

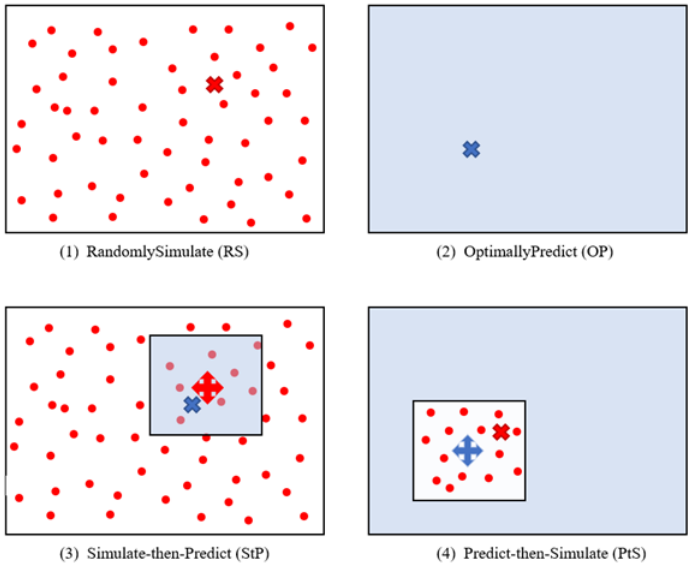

Figure 4.2: Illustration of 4 calibration strategies

entire feasible input region first, selecting the point with the least predicted error as an anchor. Next, we use our budget of r simulation runs in this narrow region to run RS close to the anchor point, and select the point with the least actual error among the simulated r points.

Figure 4.2 presents a graphic representation of the four calibration approaches discussed. The actual simulation runs are represented by red dots, and the anchor point with the least amount of error among them is the red quad. The feasible space in which OP is run is represented by blue-shaded regions, and the blue quad represents the "optimal" point—that is, the anchor point—that is discovered by the corresponding OP run. The user can specify how big the proximity region is around an anchor point. We demonstrate in our computational results how selecting different proximity region sizes affects the calibrated parameters' accuracy.

4.3.1.2 Metamodel Training

As previously indicated, the unique structure of a neural network makes it possible to embed it into an integer linear optimization model for the purpose of parameter

calibration, which is why we chose to use it as our metamodel. As a baseline architecture, we use a relatively simple neural network with an input layer of six nodes, a single hidden layer of one thousand nodes, and a multi-output layer of nine nodes. Additionally, we employ mean absolute error as the loss function during training and the rectilinear unit (ReLU) function as the activation function for hidden nodes. In order to prevent potential scaling problems, input and output are finally normalized to values between 0 and 1.

We trained two other widely used metamodels (linear regression and regression trees) and compared their predictive performance with that of neural networks in order to support our decision to use neural networks as our metamodel. Using 5000 random input parameter combinations, the simplified FluTE simulation model is used to produce a total of 5000 samples. 15% of the samples were used for testing, 15% for validation, and 70% for model training. The coefficient of determination, or R^2, served as our measure for assessing how well the metamodels predicted the future. The R^2 values for each metamodel in the training and test samples for all of the nine outputs are shown in Table 4.1. Neural network consistently performs better than the other two approaches, as can be seen. Lastly, there is almost no overfitting in the models, as evidenced by the extremely close R^2 values between the training and test samples.

	Linear Regression				Regression Tree				Neural Network			
Output	**Training R^2**	**Test R^2**	**MAE**	**MARE**	**Training R^2**	**Test R^2**	**MAE**	**MARE**	**Training R^2**	**Test R^2**	**MAE**	**MARE**
1	0.9581	0.9589	0.0469	0.2501	0.9884	0.9885	0.0239	0.1348	0.9997	0.9997	0.0039	0.0155
2	0.9614	0.9618	0.0461	0.2640	0.9897	0.9893	0.0253	0.1165	0.9995	0.9994	0.0053	0.0184
3	0.9778	0.9778	0.0357	0.2261	0.9900	0.9896	0.0252	0.1824	0.9996	0.9996	0.0045	0.0250
4	0.9743	0.9645	0.0382	0.2332	0.9871	0.9867	0.0236	0.1380	0.9998	0.9998	0.0034	0.0168
5	0.9880	0.9835	0.0262	0.1753	0.9889	0.9889	0.0217	0.1642	0.9997	0.9997	0.0041	0.0199
6	0.9689	0.9674	0.0376	0.1850	0.9683	0.9655	0.0405	0.2666	0.9941	0.9939	0.0166	0.0636
7	0.9694	0.9688	0.0370	0.1667	0.9689	0.9657	0.0380	0.1779	0.9973	0.9972	0.0112	0.0379
8	0.9408	0.9405	0.0452	0.1847	0.9510	0.9471	0.0429	0.2163	0.9782	0.9754	0.0312	0.1007
9	0.9531	0.9499	0.0396	0.1692	0.9633	0.9621	0.0408	0.1916	0.9913	0.9914	0.0185	0.0568

Table 4.1: Comparison of performance metrics for Linear Regression, Regression Tree, and Neural Network models

After we argue the selection of neural networks as the metamodeling technique, we focused on a network architecture design decision. Given that the FluTE simulation model has nine outputs, the question of whether it is preferable to train nine independent models with a single output or one single model with nine outputs (a multi-output model) emerges. The neural network architectures with

multiple outputs and one output are shown in Figure 4.3.

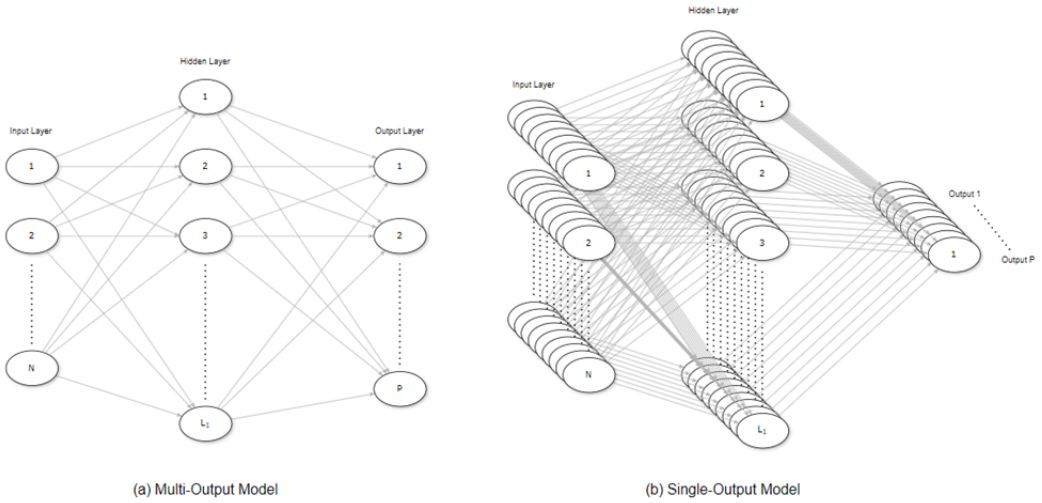

Figure 4.3: Illustration of single and multi output neural network model

We decided to use mean absolute error (MAE), mean absolute relative error (MARE), and R^2 values for each of the nine outputs for both models in the test sample in order to compare the performance of the two network architectures. The results, presented in Table 4.2, demonstrate clearly that the multi-output metamodel consistently outperforms the independent models with respect to all outputs and evaluation metrics. We think this is caused by the strong output correlations, which are best utilized when all outputs are jointly trained. From now on, we will use the multi-output metamodel as our preferred network architecture.

Lastly, we examined how the size of the training sample affected the predictive ability of the metamodel. The computational cost increases exponentially with increasing sample size because creating training samples requires simulation runs. It is therefore very important to maintain sufficient predictive power while minimizing the size of the training sample. The R^2 values in the test sample for each output at various sample sizes (n=100, 500, 1000, 2000, 5000, and 10,000) are shown in Table 4.3. The findings indicate that while 500 training samples are needed for the metamodel to perform at a level that is satisfactory ($R^2 \geq 0.97$ for all outputs), 100 training samples are insufficient to provide adequate predictive power. Moreover, it appears that the predictive power reaches a saturation point

	Single-Output Models			Multi-Output Model		
Output	**R^2**	**MAE**	**MARE**	**R^2**	**MAE**	**MARE**
1	0.9633	0.0381	0.3111	0.9997	0.0039	0.0155
2	0.9662	0.0369	0.2224	0.9994	0.0053	0.0184
3	0.9779	0.0343	0.1995	0.9996	0.0046	0.0250
4	0.9700	0.0397	0.4166	0.9998	0.0034	0.0168
5	0.9868	0.0259	0.1947	0.9997	0.0041	0.0199
6	0.9695	0.0364	0.1646	0.9939	0.0166	0.0636
7	0.9699	0.0361	0.2159	0.9972	0.0112	0.0379
8	0.9473	0.0455	0.1509	0.9754	0.0312	0.1007
9	0.9624	0.0402	0.2411	0.9914	0.0186	0.0568

Table 4.2: R^2, MAE, and MARE values (in test sample) for single and multi-output neural network models

at 1,000 samples, after which the accuracy does not significantly improve. We will carry further calibration experiments using the metamodel trained with 1000 samples in accordance with these findings.

Output	100	300	500	1000	2000	5000	10,000
1	0.95794	0.99657	0.99828	0.99894	0.99947	0.99970	0.99971
2	0.96182	0.99573	0.99806	0.99909	0.99916	0.99942	0.99942
3	0.98597	0.99524	0.99861	0.99912	0.99945	0.99959	0.99969
4	0.97886	0.99340	0.99794	0.99906	0.99962	0.99978	0.99976
5	0.99143	0.99733	0.99885	0.99919	0.99950	0.99968	0.99960
6	0.98530	0.98545	0.99253	0.99320	0.99380	0.99389	0.99462
7	0.96033	0.99153	0.99354	0.99543	0.99716	0.99716	0.99786
8	0.96065	0.96888	0.96748	0.97164	0.97150	0.97540	0.97619
9	0.93732	0.98204	0.98712	0.98837	0.99056	0.99144	0.99087

Table 4.3: R^2 values for each output by sample size

4.3.1.3 Input Parameter Calibration

We now compare the performance of the four calibration techniques—RandomlySimulate (RS), OptimallyPredict (OP), Simulate-then-Predict (StP), and Predict-then-Simulate (PtS)—introduced in previous part. For all methods (apart from OP, which doesn't require any simulation runs), we employ a budget of $r = 100$ simulation runs. The total actual error ($\Delta = \sum_{j=1}^{P} |S_j(x) - \hat{z}_j|$) is the primary metric

we use to compare various calibration techniques; however, we also report the total predicted error ($\delta = \sum_{j=1}^{P} |M_j(x) - \hat{z}_j|$) when necessary.

Comparing base strategies: The performance of the two base calibration strategies, RS and OP, is compared in the first analysis. The list of target output vectors for our experiments consists of the output vectors from 50 randomly chosen simulation runs (observations). For the i-th random observation, let $\hat{z}^i = (\hat{z}^i_1, \hat{z}^i_2, \ldots, \hat{z}^i_P)$ be the vector of target outputs. We perform both base calibration strategies for each random observation i and its corresponding target output vector $\hat{z}^i$. The input parameter vectors calibrated using RS and OP are denoted by x_i^{RS} and x_i^{OP}, respectively. Lastly, we designate the actual error of RS and OP for the observation i as $\Delta_i^{RS} = \sum_{j=1}^{P} |S_j(x_i^{RS}) - \hat{z}^i_j|$ and $\Delta_i^{OP} = \sum_{j=1}^{P} |S_j(x_i^{OP}) - \hat{z}^i_j|$. Let $\delta_i^{RS} = \sum_{j=1}^{P} |M_j(x_i^{RS}) - \hat{z}^i_j|$ and $\delta_i^{OP} = \sum_{j=1}^{P} |M_j(x_i^{OP}) - \hat{z}^i_j|$ denote the predicted error of RS and OP, respectively, for the observation i in a similar way.

The findings of experiments with 50 randomly selected target outputs are summarized in Figure 4.4. It presents the actual errors (AE) and predicted errors (PE) from 50 experiments for RS and OP, respectively, in a boxplot. In terms of the proxy/predicted error, the boxplots for PEs provide evidence that the OP model does, in fact, choose better - in fact, the best - input parameters; that is, $\delta_i^{OP} \leq \delta_i^{RS}$ holds for each individual observation i. Therefore, we can say that the OP method optimizes the proxy metric as intended. We then ask whether optimizing for a proxy metric is equivalent to optimizing for an actual metric. This time, we can clearly see that the RS method outperforms OP. The RS method chooses the point with the smallest actual error among $r = 100$ randomly sampled input combinations. This contrast highlights a crucial point: Consideration should be taken when using calibration strategies like OP, which only uses a metamodel-based optimization (of a proxy error metric). We note that selecting an input parameter combination calibrated solely based on predicted error may be suboptimal, even in cases where the trained metamodel appears to be approximating the underlying simulation model relatively accurately (in our case, the NN metamodel had test sample R^2 value greater than 0.97 for each output - see Table 4.3). As a result, we now focus on the hybrid tactics introduced.

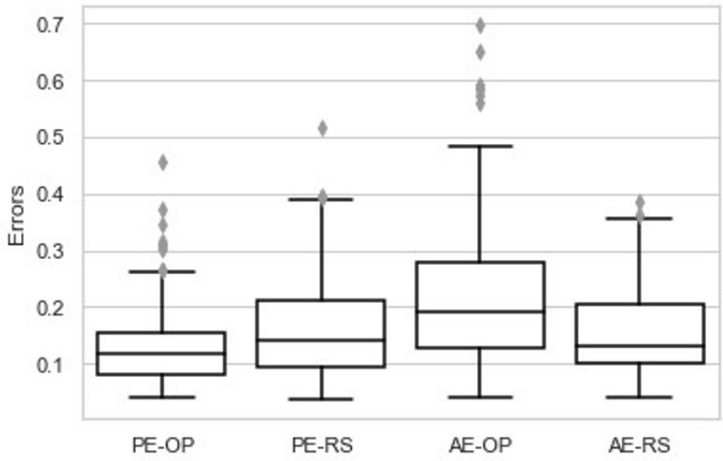

Figure 4.4: Predicted and Actual Error Boxplots for RS and OP methods

Comparing hybrid strategies: The comparison between the RS and OP base strategies indicates that the RS strategy is a better means of obtaining more precisely calibrated input parameters. However, since the RS strategy necessitates actual simulation runs, it must be noted that it is significantly more expensive (100x5 minutes vs. 30 seconds) than the OP method. "If one has a budget of $r = 100$ simulation runs for a calibration task, how should they be used?" becomes an important question as a result. We will next evaluate the effectiveness of two hybrid calibration strategies previously discussed, assuming that we have a trained metamodel on hand and a fixed budget of $r = 100$. These strategies are (iii) Simulate-then-Predict (StP), (iv) Predict-then-Simulate (PtS).

An initial anchor point and a proximity range surrounding that anchor point are necessary for both hybrid strategies. StP employs the OP strategy to look for better solutions close to the improved anchor point after using its budget of r simulation runs with the RS strategy to create a better anchor point. PtS, on the other hand, uses its budget of r simulation runs to find a better point within a radius of that anchor point via RS strategy after first finding an anchor point via OP strategy (which is not always the best one, as the results in the previous subsection demonstrated).

Finding an anchor point is simple for both strategies (you just need to run RS or OP, respectively). However, user input is needed to define the anchor point's

proximity range for additional search. The anchor point loses significance as the proximity range increases. The search spaces for different proximity ranges in a two-dimensional PtS strategy parameter space are shown in Figure 4.5. The point from OP with the smallest predicted error is indicated by the point with the anchor symbol. In the other images, the proximity range ($R \in \frac{1}{50}, \frac{1}{20}, \frac{1}{10}, \frac{1}{5}$) is increased, indicating a wider area around the anchor point. In the following series of experiments, $R = \frac{1}{10}$ is used, we set the vicinity of the anchor point in each input dimension as follows: $x_i \in \text{AnchorPoint}_i \pm \frac{1}{10} \times (\text{ub}_i - \text{lb}_i)$

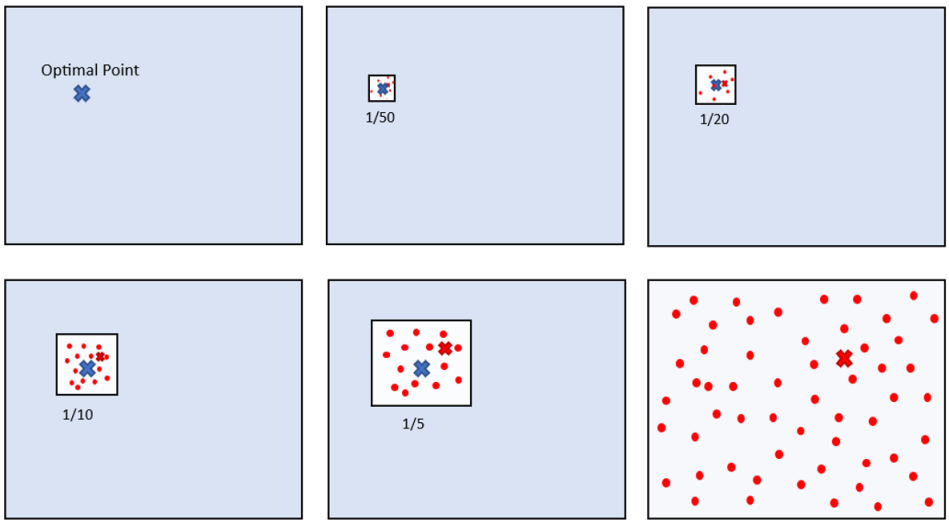

Figure 4.5: Illustration of PtS strategy with different proximity ranges

The actual error distributions from 50 experiments with randomly chosen target outputs for each of the four strategies are displayed in boxplots in Figure 4.6. It is clear that the PtS strategy performs noticeably better than the other three strategies. All methods' performances can be arranged in the following order: $PtS > RS > StP > OP$. These findings demonstrate the benefit of making practical use of scarce computational resources. To be more precise, while blindly sampling the entire input space results in significantly lower calibration accuracy, using OP to narrow it down for a more computationally intensive search (i.e., random simulation runs) yields significantly higher calibration accuracy at relatively little additional computational cost. This is in contrast to solely relying on a metamodel-based calibration strategy, which yields the worst performance. Lastly, the fact that PtS outperforms StP offers crucial evidence that the base

strategies' execution order matters. The best results are obtained when an RS is used to identify the optimal point after an OP is used to reduce the search space.

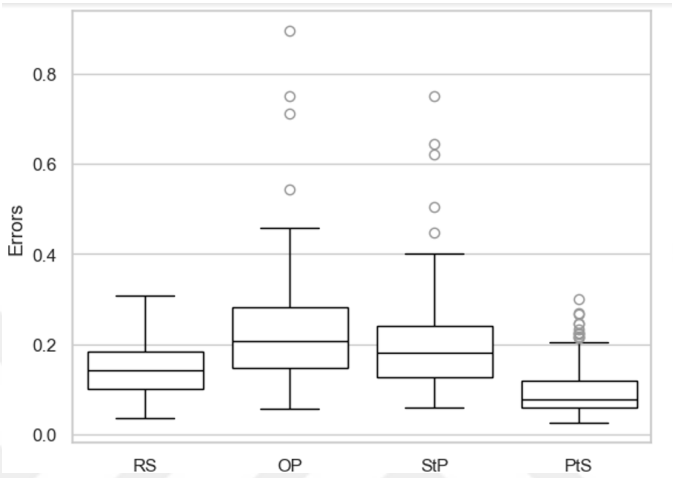

Figure 4.6: Actual Error distributions for 4 calibration strategies

Effect of proximity ranges in calibration accuracy: We set the proximity range around the anchor point in the previous experiment to $\frac{1}{10}$ of the initial range for each input parameter. Since this was a purely subjective decision, we examine the effects of selecting alternative proximity range values in the last set of experiments. As previously demonstrated in Figure 4.6, the rectangular borders are created by multiplying the original parameter range by $R \in \left\{\frac{1}{5}, \frac{1}{10}, \frac{1}{20}\right\}$, and then calculating the distance from the anchor point in each parameter dimension. With six input parameters, the search space was thereby reduced by 5^6, 10^6, and 20^6 times, respectively. Now, we will examine the PtS strategy's performance (i.e., actual error distributions) for various proximity range values; $R \in \left\{\frac{1}{1}, \frac{1}{5}, \frac{1}{10}, \frac{1}{20}, \frac{1}{50}, \frac{1}{1000}\right\}$. Observe that $R = \frac{1}{1}$ represents the entire input domain, while $R = \frac{1}{1000}$ roughly corresponds to the anchor point. As a result, RS and OP strategies are represented by PtS with $R = \frac{1}{1000}$ and PtS with $R = \frac{1}{1}$, respectively.

For the cases of $R = \frac{1}{1000}$ (OP), $\frac{1}{50}$, $\frac{1}{20}$, $\frac{1}{10}$, $\frac{1}{5}$, and $\frac{1}{1}$ (RS), the actual error distributions are displayed in Figure 4.7 from left to right. The boxplots and median error values demonstrate how the performance of the PtS strategy increases as the proximity range around the anchor point grows until approximately $R = \frac{1}{20}$, and after that point it begins to decline.

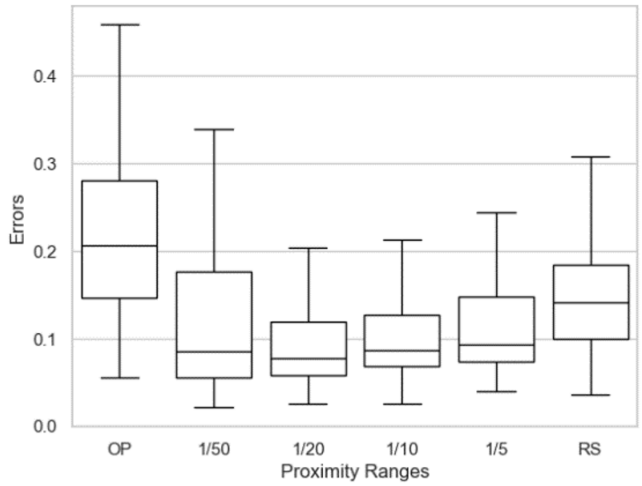

Figure 4.7: Error distributions for different ranges in PtS method

4.3.2 Case Study 2: Assortment Optimization Problem

4.3.2.1 Problem Definition and Customization of Methodology

We have a real-life FMCG system at hand such that; $z = F(x)$, with $x = (x_1, x_2, \ldots, x_N)$ as the vector of inputs (assortment information) and z as the system's output (GSV). The problem can be defined as a profit maximization problem by changing the assortments in stores, which means finding an assortment for the highest sales value. The optimization model can be formulated as follows (assume minimization problem);

$$x^* = \arg\max_{x \in \mathcal{X}} F(x)$$

where the feasible assortments are represented by $\mathcal{X}$. Similar to the general problem definition, mathematical expression of the system F is not possible due to high complexity between the assortments and sales value. So, a neural network model is used to approximate the relationship between the inputs and the output. Again, let $L(x)$ be the neural network model that efficiently approximates $F(x)$ globally, i.e.,

$$L(x) \approx F(x), \forall x \in X$$

Let $\tilde{x}^*$ represent the optimal assortment derived from the problem:

$$\tilde{x}^* = \arg\max_{x \in X} F(x)$$

Then, $\tilde{x}^*$ can be used to approximate x^*; thus, $\tilde{x}^* \approx x^*$.

Change in the Base Model

Considering this problem, following parameters and decision variables are added to the model;

Additional Parameters		
$\hat{x}_i$	Current assortment information	$\forall i = 1, ..., N$
c	Number of changes allowed from the current assortment	

Additional Decision Variables		
δ_i	Difference between the optimal assortment and current assortment	$\forall i = 1, ..., N$

The objective $\mathcal{F}$ is defined as; *maximize* z (Notice P=1)

Following additional constraints are added to the model;

$$\delta_i = \hat{x}_i - x_i \qquad \forall i = 1, \ldots, N | \hat{x}_i = 1 \tag{13.1}$$

$$\sum_{i|\hat{x}_i=1} \delta_i \leq c \tag{13.2}$$

$$\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} \hat{x}_i \tag{13.3}$$

$$x_i \in \{0, 1\} \qquad \forall i = 1, \ldots, N \tag{13.4}$$

Constraint 13.1 calculates the difference between the optimal and current assortment by only considering products in the current assortment. Constraint 13.2 ensures that the number of changes from current assortment is less than or equal to specific user defined number c (some computational experiments in later sections will take this as equal to c). Lastly, 13.3 ensures number of products in optimal assortment found is the same as the current assortment.

4.3.2.2 Neural Network Training

For this case study, as we had available real-life data, all data points were used in training the neural network. With the dataset, a pre-trained neural network was also provided which had 2 hidden layers with 400 and 40 nodes respectively. %20 of the dataset was used for testing, %20 for validation, %60 for model training. Similarly to other case study R^2 was used for main accuracy measure. The R^2 values for training, validation and testing were found as 0.81658, 0.7597, and 0.8285 respectively which shows sufficient approximation performance.

4.3.2.3 Assortment Results

As we do not have a simulation model for this case study, the potential computational experiments that can be done was limited and making comparisons between predicted profit increases within different times and actual changes was not possible. However, a previous study done with the same dataset and base methodology reports a %14.31 GSV increase in a pilot study for one month which closely matches the predicted increase that ranges from %12 to %16, for multiple time periods and assortments, which we have found utilizing the suggested methodology.

4.4 Discussion

By inspecting the computational results, we can say that the proposed methodology provides a successful approach to solve problems in real-life systems with hidden inner structures. Although the optimization over neural networks provides an environment to make complex inferences about the real-world systems with nearly negligible computational cost, the reliability of these results can be further improved as it can also be seen from the hybrid approaches utilized in simulation case study. As it can also be seen from the Figure 4.8, the point predictions can change for a single instance according to the approximation model

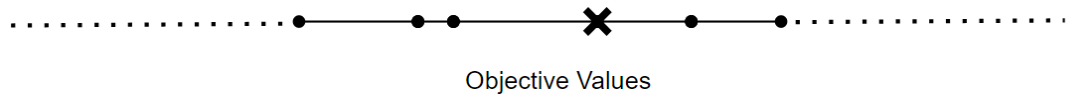

Figure 4.8: Representation for change in predictions with different approximation models for a single instance

used which can be misleading. The cross represents the objective from the main neural network model and dots represent the predictions from different approximation models (e.g., dropout networks) which will be defined in the next section. According to the results, we can say that there is a ground for improvement to make the approximations more robust and reliable by employing more complex approaches and exploiting the structure of our choice of approximation model, neural networks. In the next chapter, we will introduce an improved approach that utilizes dropout neural networks to alleviate the problem of reliability in neural network predictions.

Chapter 5

Estimation Error Constrained Optimization of Point Predictions

5.1 Problem Definition

Following the definitions on previous chapter, we have $L(x)$, a machine learning model that efficiently approximates $R(x)$, a real life system, i.e.,

$$L(x) \approx R(x), \quad \forall x \in X$$

Optimal parameter combination found $\tilde{x}^*$ derived from the problem:

$$\tilde{x}^* = \arg\min_{x \in X} L(x)$$

That should reasonably approximate optimal interventions in a real-world system. However, as the computational results have shown the reliability of predictions varies according to the training procedure and value of data points. Hence, we also want to define a measure of estimation error on approximation which gives an approximation range rather than point prediction. One possible measure of estimation error can be defined as the difference between predictions. Let's assume multiple trained models for the system besides the main model, namely L_r, $r_1, ..., R$ such that;

$$L_r(x) \approx R(x), \quad \forall x \in X, \forall r \in R$$

We can redefine the problem as;

$$\tilde{x}^* = \arg\min_{x \in X} L(x) \quad | \quad L_r(x) - L_{r'}(x) \leq \epsilon \quad \forall r, r' \in R$$

which yields stronger approximation results by restricting predictions to be similar to each other by value ϵ.

5.2 Methodology

5.2.1 Dropout Neural Networks

As mentioned in Section 2.2, dropout networks are commonly used in estimating uncertainty in neural network predictions as they ensure predictions are not strongly dependent on any small part of the network structure.

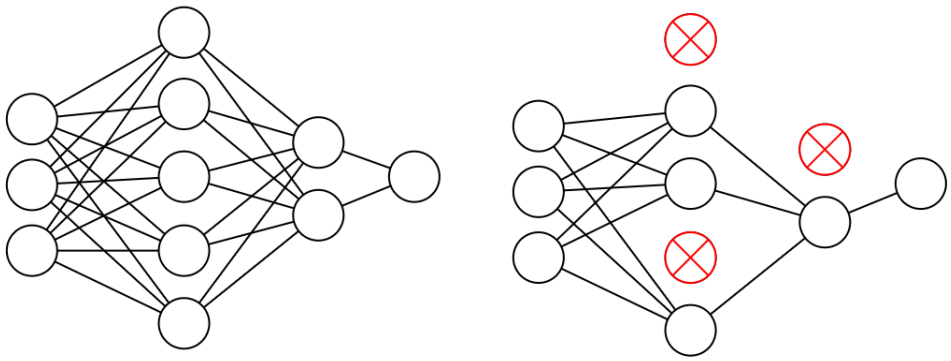

Figure 5.1: Dropout Neural Network Structure

An example of dropout neural networks can be seen in Figure 5.1. For a system with 3 inputs and 1 output, a neural network with two hidden layers including 5 and 2 nodes respectively is trained. After applying dropouts, some of the nodes in hidden layers are randomly deactivated/silenced and output prediction is made only using the active nodes. To normalize the prediction values, remaining node values are multiplied with $1 + d$, where d is the dropout rate, for example, $1 + \frac{3}{7}$ in this case. For instance, let dropout rate = d, the calculation for first node in

layer 2 becomes;

$$y_{21}^r = (1+d) * \sigma(b_{12}^r + w_{111}^r y_{11}^r + w_{121}^r y_{12}^r + \cdots + w_{1L_11}^r y_{1L_1}^r)$$

where w_{ksl}^r and b_{kl}^r becomes 0 if node l in layer k is dropped in dropout neural network r.

We decided on randomly dropping the nodes in the network. To make calculations easier, rather than changing the values of weights and biases, indicator parameters were generated to represent if the node is active or dropped out and how much they change the node output. Given the number of dropout networks and dropout rate, following parameter is defined;

$$I_{kl}^r = \begin{cases} 0 & \text{if node l in layer k is dropped out for dropout network r} \\ & \forall k = 0, ..., K, \forall l = 1, ..., L_k, \forall r = 1, ..., R \\ 1+d & \text{otherwise} \end{cases}$$

Using this parameter, the node outputs can be calculated without defining network weights and biases for each dropout. Hence, following the above example the calculation for first node in layer 2 for dropout network r becomes;

$$y_{21}^r = I_{21}^r * \sigma(b_1^2 + w_{11}^1 y_1^1 + w_{21}^1 y_2^1 + \cdots + w_{L_11}^1 y_{L_1}^1)$$

5.2.2 MILP Model with Dropout Networks

As mentioned before, we now want to have an estimation error measure that is calculated with the optimal input parameter configuration. To ensure the reliability of the optimal result, after generating the indicator parameters for dropout networks, they are similarly used inside the MILP model to make multiple predictions. Then, their values are compared inside this mathematical model to find a result that have similar predictions for these different dropout networks. The parameters and decision variables are defined similarly to base model (i.e., point prediction optimization model) with most of them having an additional index for different dropout networks.

Parameters		
ϵ	Parameter for estimation error constraint	
R	Number of dropout ANNs+1 (original model)	
M	An arbitrarily large number, e.g. $M = 10^6$	
N	Number of input parameters	
P	Number of outputs	
K	Number of hidden layers	
I^r_{ks}	0 if node s in layer k is dropped in network r, 1+d otherwise	$\forall k = 0, ..., K, \forall s = 1, ..., L_k,$ $\forall r = 0, ..., R$
L_k	Number of nodes in the (hidden) layer k $L_0 = N$ and $L_{(K+1)} = P$	$\forall k = 0, ..., (K + 1)$
w^k_{st}	Weight between node s in layer k and node t in layer (k+1) in dropout r	$\forall k = 0, ..., K, \forall s = 1, ..., L_k,$ $\forall t = 1, ..., L_k$
b^k_t	Bias value for node t in layer k for dropout r	$\forall k = 1, ..., K + 1, \forall t = 1, ..., L_k$
lb_i	Lower bound value for i^{th} input parameter	$\forall i = 1, ..., N$
ub_i	Upper bound value for i^{th} input parameter	$\forall i = 1, ..., N$

Decision Variables		
x_i	Value of i^{th} input parameter	$\forall i = 1, ..., N$
y^r_{kt}	Value of node t in hidden layer k in dropout r	$\forall k = 0, ..., (K + 1), \forall r = 0, ..., R$
u^r_{kt}	Binary variable indicating if node t in layer k has a nonzero value in dropout r	$\forall k = 0, ..., K,\ \forall t = 1, ..., L_k, \forall r = 0, ..., R$
z^r_j	Value of j^{th} output for dropout r	$\forall j = 1, ..., P, \forall r = 0, ..., R$

$$\min \quad \mathcal{F}(x) \tag{1}$$

$$\text{s.t.} \quad y^r_{1t} \geq I^r_{1l}(b^1_t + \sum_{i=1}^{N} w^0_{it} x_i) \qquad \forall t = 1, ..., L_1, \forall r = 0, ..., R \tag{2}$$

$$y^r_{1t} \leq I^r_{1l}(b^1_t + \sum_{i=1}^{N} w^1_t x_i + M(1 - u^r_{1t})) \qquad \forall t = 1, ..., L_1, \forall r = 0, ..., R \tag{3}$$

$$y^r_{1t} \leq I^r_{1l} M u^r_{1t} \qquad \forall t = 1, ..., L_1, \forall r = 0, ..., R \tag{4}$$

$$y^r_{k+1,t} \geq I^r_{kl}(b^{k+1}_t + \sum_{s=1}^{L_k} w^k_{st} y^k_s) \qquad \forall t = 1, ..., L_k, \forall k = 1, ..., K - 1$$
$$\forall r = 0, ..., R \tag{5}$$

$$y^r_{k+1,t} \geq I^r_{kl}(b^{k+1}_t + \sum_{s=1}^{L_k} w^k_{st} y^r_{ks} + M(1 - u^r_{k+1,t})) \qquad \forall t = 1, ..., L_k, \forall k = 1, ..., K - 1$$
$$\forall r = 0, ..., R \tag{6}$$

$$y^r_{k+1,t} \leq I^r_{kl} M u^r_{kt} \qquad \forall t = 1, \ldots, L_k, \forall k = 1, ..., K \quad \forall r = 0, ..., R \tag{7}$$

$$z^r_j = b^{K+1}_j + \sum_{s=1}^{L_K} w^K_{sj} y^r_{Ks} \qquad \forall j = 1, ..., P, \forall r = 0, ..., R \tag{8}$$

$$lb_i \leq x_i \leq ub_i \qquad \forall i = 1, ..., N \tag{9}$$

$$y^r_{kt} \geq 0 \qquad \forall k = 1, \ldots, K, \forall t = 1, ..., L_k \quad \forall r = 0, ..., R \tag{10}$$

$$u^r_{kt} \in \{0, 1\} \qquad \forall k = 1, ..., K, \forall t = 1, ..., L_k \quad \forall r = 0, ..., R \tag{11}$$

$$z^r_j \text{unrestricted} \qquad \forall j = 1, ..., P, \forall r = 0, ..., R \tag{12}$$

$$x \in \mathcal{X} \tag{13}$$

$$z^r_j \leq (1 + \epsilon) z^0_j \qquad \forall j = 1, ..., P, \forall r = 0, ..., R \tag{14.1}$$

$$z^r_j \geq (1 - \epsilon) z^0_j \qquad \forall j = 1, ..., P, \forall r = 0, ..., R \tag{14.2}$$

Constraints (3)-(7) are changed according to the equation provided on the previous section. Additional indicator multiplications are added to force node values 0 if nodes are dropped, or proportionally increase the node outputs if not. As the nodes in input and output layers remain the same, constraint (8) stays same. Constraints (14.1) and (14.2) enforce dropout network predictions to be in a vicinity of the main network prediction (indicated by $r = 0$) by ϵ. The estimation error constraints can be defined differently according to the specific problem structure.

5.3 Computational Experiments

5.3.1 Case Study 1: FluTE Parameter Calibration

5.3.1.1 Customization of Methodology

To start with, lets first write the main model without extending. The additional parameters and decision variables remain the same as the previous chapter. The objective $\mathcal{F}$ is defined as; *minimize* $\sum_{j=1}^{P} \delta_j$.

Two additional constraints are similarly defined as;

$$\delta_j \geq \hat{z}_j - z_j^0 \qquad \forall j = 1, \ldots, P \tag{13.1}$$

$$\delta_j \geq z_j^0 - \hat{z}_j \qquad \forall j = 1, \ldots, P \tag{13.2}$$

with an index 0 on decision variable z to indicate the predicted output from the main neural network. We have done our first set of computational analysis with previously defined estimation error constraints (14.1) and (14.2).

Then, due to problem specifications we decided on changing the estimation error constraints (14.1) and (14.2), as they were controlling vicinity of the dropout values with respect to the main predicted output and we similarly calculate the difference between the target and predicted output. Hence, we wrote the constraints by making the calculations by directly considering the target value. A new decision variable set is added to the model;

Additional Decision Variables		
γ_j^r	Difference between the target value and the predicted dropout output	$\forall j = 1, ..., P, \forall r = 1, ..., R$

And the estimation error constraints are defined as follows;

$$\gamma_j^r \geq \hat{z}_j - z_j^r \qquad \forall j = 1, \ldots, P, \forall r = 1, \ldots, R \tag{14.1}$$

$$\gamma_j^r \geq z_j^r - \hat{z}_j \qquad \forall j = 1, \ldots, P, \forall r = 1, \ldots, R \tag{14.2}$$

$$\sum_{j=1}^{P} \gamma_j^r \leq \epsilon \qquad \forall r = 1, \ldots, R \tag{14.3}$$

in which we constrained the maximum total difference between the target output and the dropout predicted outputs.

5.3.1.2 Dropout objectives vs maximum difference limits

As mentioned before, we are now utilizing the optimization model with estimation error constraints defined as (14.1), (14.2) and (14.3), which ensure the maximum difference between the target output and the predicted output from dropout networks is less than some ϵ value. With these constraints, the optimal result found will provide that the difference to target output will still be less than some value even for the worst case scenario. By taking the number of dropouts 5 and dropout rate 0.2, the dropout objectives are recorded with different maximum difference/variance limits. The results can be seen in Figure 5.2. As we expected, we can see an increase in the predicted objective which, in this case, is the total deviation from the target outcome, while the bound on variance becomes tighter. Hence, our confidence on the objective found also becomes higher which should show an effect in actual error values by approximating the objectives with higher accuracy.

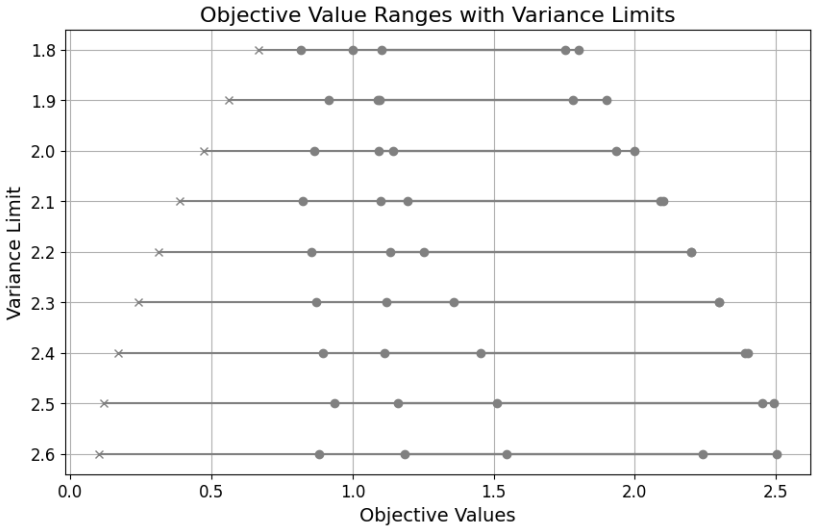

Figure 5.2: Dropout values with different maximum difference limits

5.3.1.3 Dropout objectives vs dropout rates

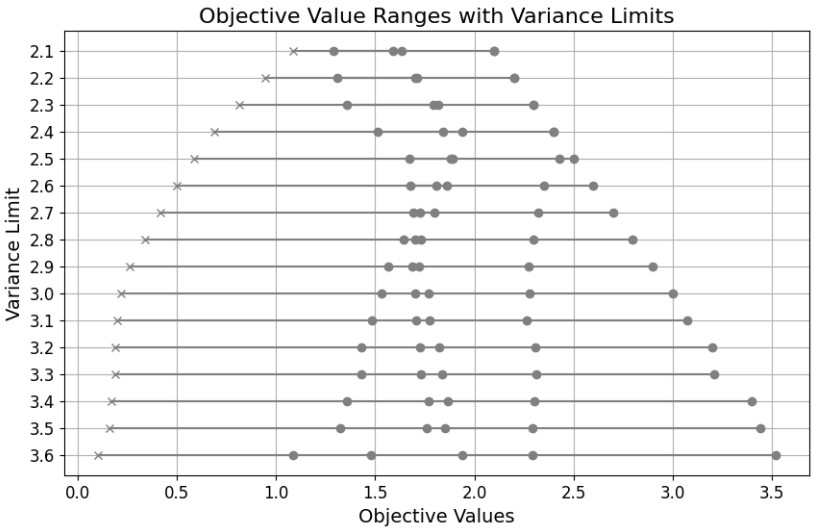

Figure 5.3: Dropout values with different maximum difference limits and dropout rates

For the same target value and number of dropouts, the objective and dropout objectives were recorded for two different dropout rates; 0.2 and 0.3. The objective values for dropout rate 0.2 and 0.3 can be seen in Figure 5.2 and Figure 5.3 respectively. As expected, with higher dropout rate the maximum difference found also started higher. However, the lowest variance limit both had a feasible solution was also close to each other, which can result in a more reliable approximated point with higher dropout rate. To analyze the effect of dropout rate in measuring uncertainty further experiments are needed that takes the actual errors into account.

5.3.1.4 Dropout objectives vs number of dropouts

The effect of number of dropouts was tested by recording the dropout objective with maximum value for one targets' solution by randomly sampling each dropout 100 times. The results can be seen in Figure 5.4. As it can be seen in the figure, the maximum objectives with generated random dropout values have higher variance when dropout number is smaller as we expected. As the number of dropouts

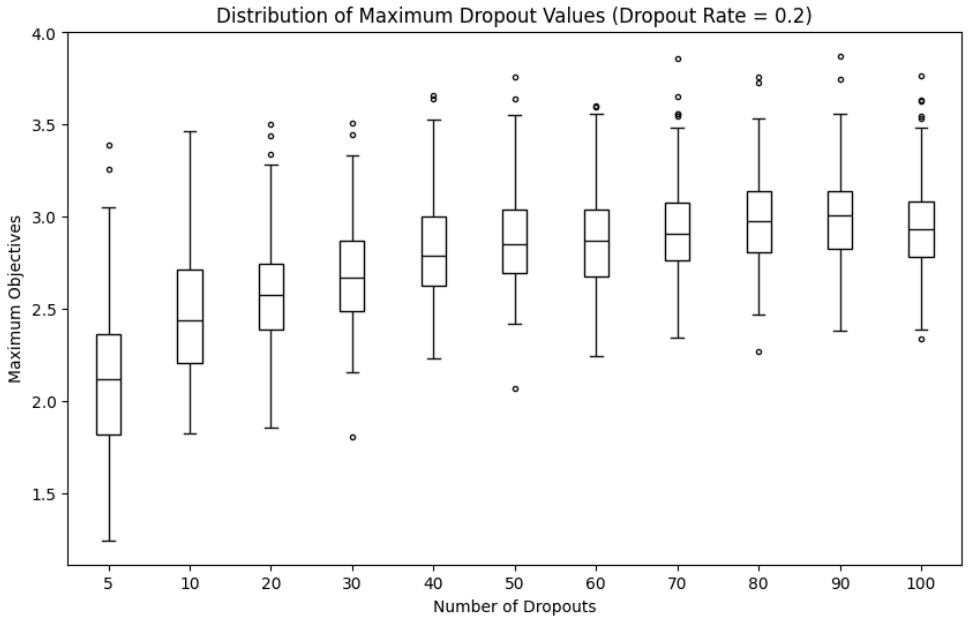

Figure 5.4: Distribution of maximum dropout objectives with different number of dropout networks

increase up to 40-50 the distribution of maximum objectives starts to converge. Which means after 40 dropout using more networks will become unnecessary and only increase the computational cost. Before 40 dropout, the reliability of the solutions may not be optimal as the dropout networks may underestimate the error.

5.3.1.5 Dropout objectives vs training methods

We tried to see the effect of how the training samples and target output are selected by using two methods. We first splitted the dataset according to the parameter values so that one part only had points with each parameter being less than the %60 of its maximum value. The training of the neural network was done by using data points from this part only. Then, two target points were selected, one from the first split and other from outside of it. The representations can be seen in Figure 5.5. The dots represent the values of training samples and cross represents the target value.

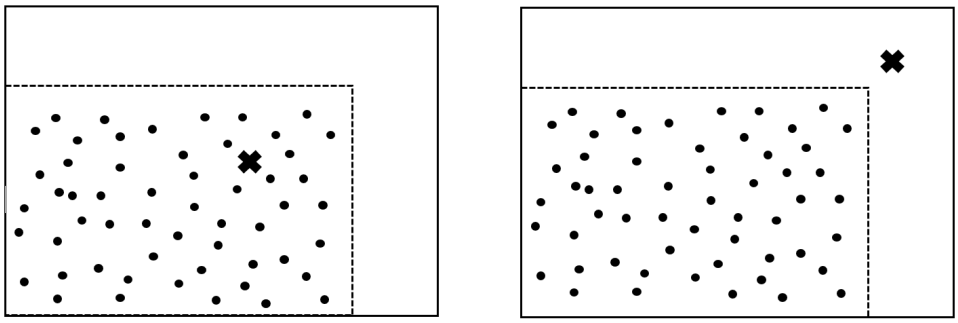

Figure 5.5: Representation for target value selection

The maximum dropout objective values with different number of dropout networks are presented in Figure 5.6. One can see the direct effect of how training samples are selected and underfitting for data points in a specific region. As the targets and optimal input parameters are found outside of the trained region, their error clearly becomes higher compared to a point we know is in a region we learned well. Also, as the number of dropouts increase, the gap between the error values for the two method becomes bigger as we expect due to more reliable measurement of estimation error in predictions.

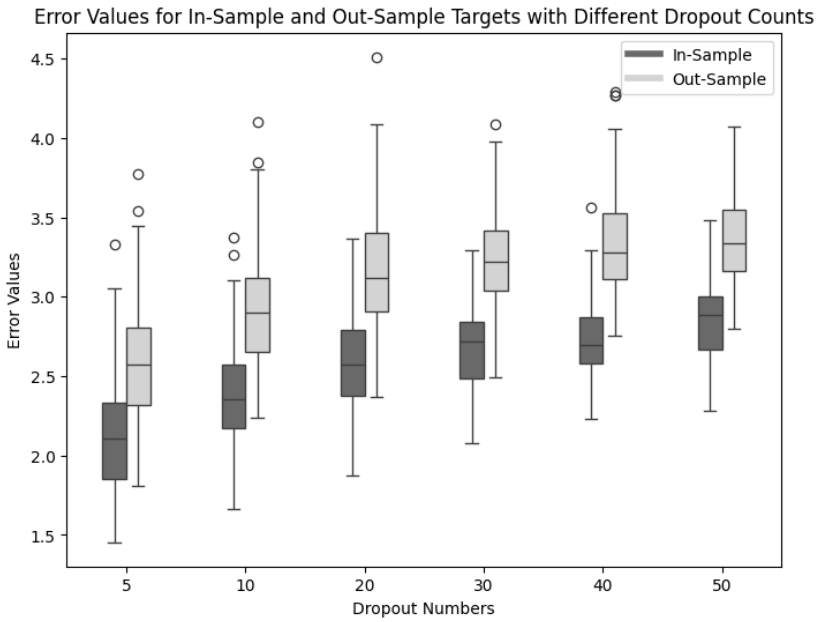

Figure 5.6: Distribution of objectives with different training methods and number of dropout networks

5.3.2 Case Study 2: Assortment Optimization Problem

5.3.2.1 Customization of Methodology

The additional parameters and decision variables remain same with the previous chapter, and the problem specific constraints are similarly added to the model. The objective $\mathcal{F}$ is again defined as; *maximize* z.

We utilized two set of estimation error constraints. The first set is as defined in the generalized optimization model;

$$z_j^r \leq (1+\epsilon) z_j^0 \qquad \forall j = 1, ..., P, \forall r = 0, ..., R \qquad (14.1)$$

$$z_j^r \geq (1-\epsilon) z_j^0 \qquad \forall j = 1, ..., P, \forall r = 0, ..., R \qquad (14.2)$$

which constraints the predicted values from dropout networks according to the predicted value from main network by some proportion ϵ.

Then, we used the estimation error constraints defined as;

$$\epsilon \geq z^r - z^0 \qquad \forall r = 1, \ldots, R \qquad (14.1)$$

$$\epsilon \geq z^0 - z^r \qquad \forall r = 1, \ldots, R \qquad (14.2)$$

in which the difference between the dropout prediction and the main network prediction is constrained by a value.

Heuristics: We also developed an heuristic approach by using the problem specific constraint to further improve the computational efficiency. We tried making one change to the current assortment with each iteration and using greedy approach to fix the initial change to continue up to allowed number. The explanation can be seen in Algorithm 1 and Algorithm 2. The last element of the output lists of the Algorithm 1 gives the optimal assortment and the optimal objective respectively.

Algorithm 1 Heuristic Algorithm for Maximizing Objective

Input: Initial assortment $\hat{x}$, number of changes allowed d, variance limit ϵ, Dropout neural networks L_r
Output: Assortment updates and Best objectives
Initialize Assortment_updates $\leftarrow [\hat{x}, \hat{x}]$
Initialize $x_{iter} \leftarrow \hat{x}$
Initialize Best_Objectives $\leftarrow []$
for $i = 1$ to d **do**
    updated_lists $\leftarrow$ enumerate_one_change(x_{iter})
    Initialize all_preds $\leftarrow []$
    Initialize all_ranges $\leftarrow []$
    for each $x^{updated}$ in updated_lists **do**
        if updated_list $\neq$ iterative_list_updates$[-2]$ **then**
            $x^r \leftarrow L^r(x^{updated}) \forall r = 0, ..., R$
            max_range $\leftarrow \max_{r \in R}(|x^0 - x^r|)$
            Append max_range to all_ranges
            Append $[x^r, \forall r \in R]$ to all_preds
        end if
    end for
    filtered_indices $\leftarrow \{i \mid \text{all\_ranges}[i] \leq \epsilon\}$
    max_index $\leftarrow \arg\max_{i \in \text{filtered\_indices}} \text{all\_preds}[i]$
    Append updated_lists[max_index] to Assortment_updates
    Append [all_preds[max_index], all_ranges[max_index]] to Best_Objectives
    $x_{iter} \leftarrow$ updated_lists[max_index]
end for
return Assortment_updates, Best_Objectives

Algorithm 2 Enumerate One Change

Input: Original assortment $\hat{x}$
Output: New assortments with one change
Initialize new_assortments $\leftarrow$ []
one_indexes $\leftarrow \{i \mid \hat{x}[i] = 1\}$
zero_indexes $\leftarrow \{i \mid \hat{x}[i] = 0\}$
for each value$_1$ in one_indexes **do**
    new_assortment $\leftarrow$ (original_list)
    if new_assortment[value$_1$] = 1 **then**
        new_assortment[value$_1$] = 0
        for each value$_0$ in zero_indexes **do**
            updated_assortment $\leftarrow$ new_assortment
            if updated_assortment[value$_0$] = 0 **then**
                updated_assortment[value$_0$] = 1
                Append updated_assortment to new_assortments
            end if
        end for
    end if
end for
return new_assortments

5.3.2.2 Objective vs Variance Limit vs Allowed Assortment Change

To see the effect of variance limit and allowed assortment changes with the initial estimation error constraints, we reported the dropout network predictions with different variance limits in each allowed change groups. Note that variance limit 1 represents no constraint for estimation error, hence, utilizing the base model from previous chapter while also recording the dropout network predictions for the optimal point.

As it can be seen from the Figure 5.7, increasing the number of changes allowed from the current assortment also increases the predicted revenue which was an expected result intuitively. With the decreasing variance limits we lose some value from our objective, however, it also makes the dropout predictions closer. Hence, it gives a point where we approximate with more confidence. Some outlier values can also be seen in the figure due to the optimization time limit and the stopping gap. In general, increase in objectives with higher assortment charge

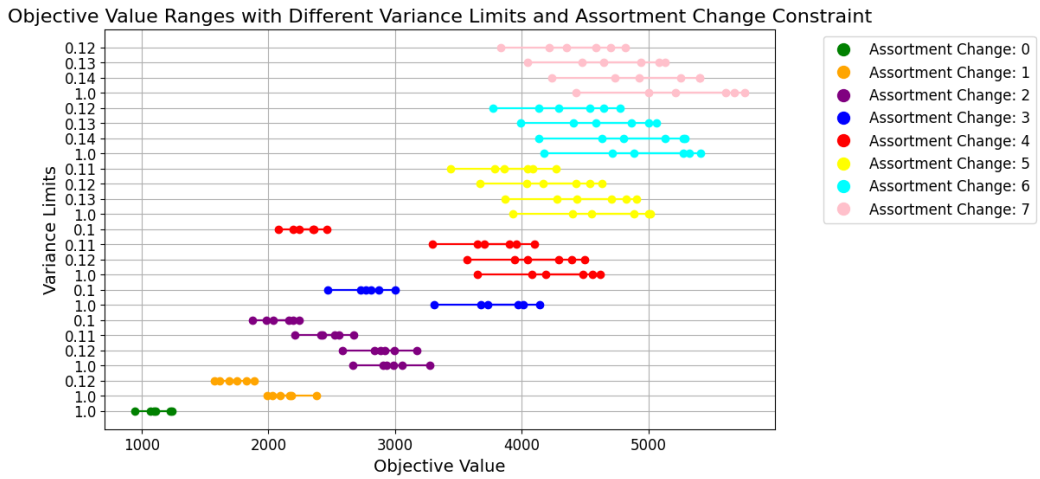

Figure 5.7: Objective changes with different variance limits and allowed assortment change

and decreasing objectives with higher variance limit is observed.

5.3.2.3 Objective vs Range Limit vs Allowed Assortment Change

Similar experiments were made with another measure of estimation error. Similar to checking variance between the dropout values proportionally, now we checked using the range values. As it was mentioned before, the results gave similar structure to the method with initial variance limit. From Figure 5.8, with the increase in allowed assortment change, the objectives increase. Also, looser limit on the range between approximations increased the dropout values but also decreased the confidence. However, compared to 5.7 the increase in the objective with allowed assortment change is less evident as the estimation error limit is defined numerically, not proportionally. Outliers are again caused by the gap not closed due to time limit. Please note that the target values are different from the Figure 5.7, hence, there is a difference in objective values.

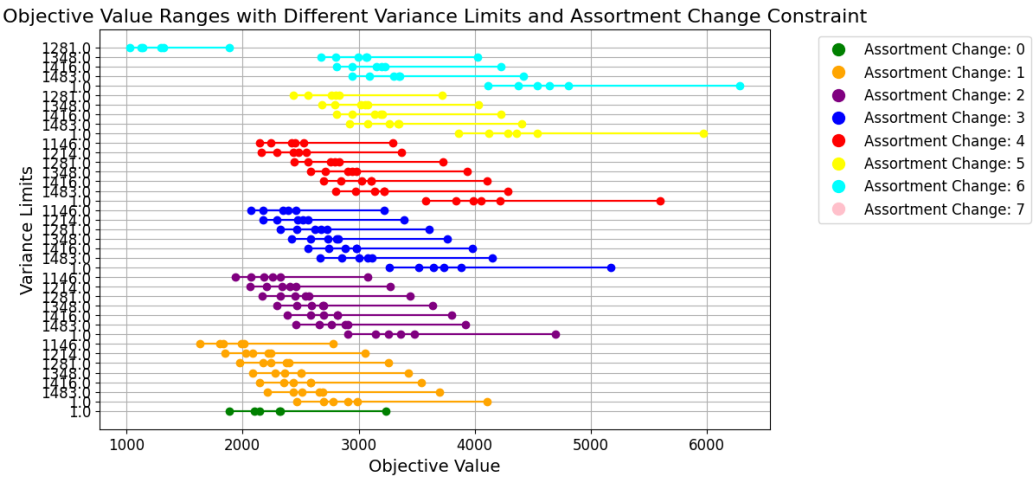

Figure 5.8: Objective changes with different range limits and allowed assortment change

5.3.2.4 Heuristics vs Optimization

To compare the heuristic and optimization model's results we first recorded the values with no estimation error constraints, hence the base model for optimization and iterative greedy assortment changes. As it can be seen from the Figure 5.9, two methods gave approximately same values in each instance with heuristic having a computational benefit. Heuristic approach was only needed to run one for the 15 allowed assortment and other results were also returned as we save the list for each iteration. On the other hand, optimization was run for each allowed change parameter while also having larger computational time. In accordance to this result, we tried the methods for different variance limits. Heuristic approach still maintained it's computational benefit but the objective values found from optimization approach was slightly better. Also, after assortment change 3 or 4, for most of the target values heuristic approach returned empty set, infeasible, as we were losing many feasible points after each iteration. Hence, we can say that heuristic approach may offer some benefit with relatively less allowed assortments changes but the results should also be tested on real-life to compare the reliability of them. For higher allowed assortment changes, or for more complex systems, the optimization approach with estimation error constraint performs better.

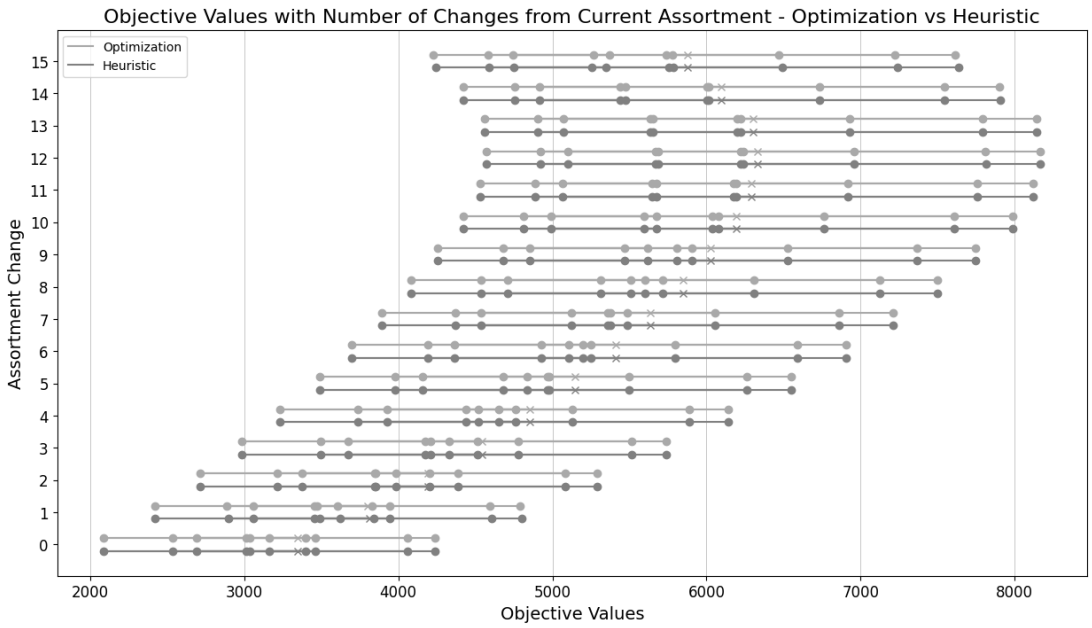

Figure 5.9: Objective Changes for Different Allowed Assortment Changes with No Variance Limit (Optimization vs Heuristic)

5.4 Decomposible Structure and Extended Model

Notice that the defined decision variables in the optimization model mostly have r index and constraints are all defined for each dropout network with nearly no joint usage in one constraint. Except the measure of estimation error constraints (14.1) and (14.2), what binds sets of constraints defined for each dropout network are only the input variables. This implies a potential decomposable structure for improving computational efficiency. However, due to problem specific constraint set (13) decomposed models may change. Hence, we will only present the extended model in this section.

The change in the model structure can be seen in Figure 5.10. As mentioned before, ignoring the constraints (14.1) and (14.2), the only binding constraints in the optimization model are (2),(3), and (9) with the shared decision variables x_i. Hence to extend the model, the x_i variables are copied across for each dropout network, redefining them as;

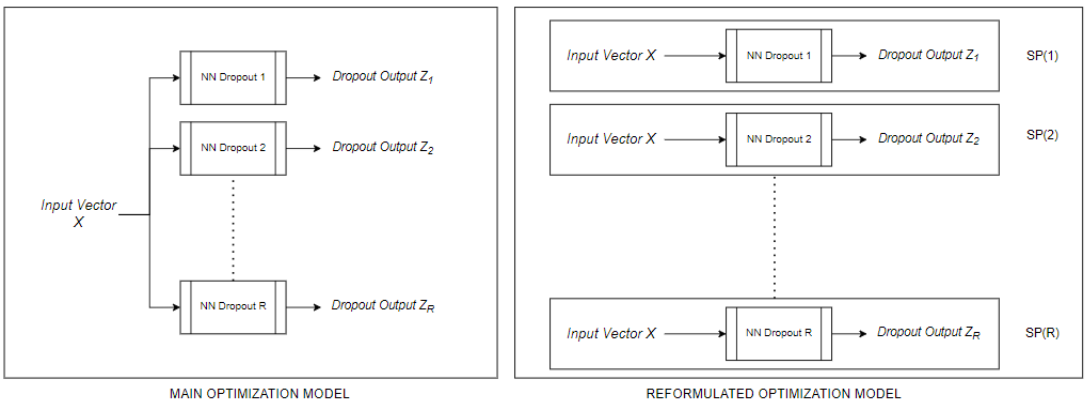

Figure 5.10: Structure of the main model and reformulated model

x_i^r : Value of input parameter i=1,...,N for dropout network r=1,...,R

After copying the decision variables, the model constraints change as;

$$y_1 t^r \geq I_{1l}^r (b_t^1 + \sum_{i=1}^{N} w_{it}^0 x_i^r) \qquad \forall t = 1, ..., L_1, \forall r = 1, ..., R \qquad (2)$$

$$y_{1t}^r \leq I_{1l}^r (b_t^1 + \sum_{i=1}^{N} w_t^1 x_i^r + M(1 - u_{1t}^r)) \qquad \forall t = 1, ..., L_1, \forall r = 1, ..., R \qquad (3)$$

$$lb_i \leq x_i^r \leq ub_i \qquad \forall i = 1, ..., N, \forall r = 1, ..., R \qquad (9)$$

Remaining constraints stay same. An additional constraint set is defined to ensure the equality of input parameters for each dropout;

$$x_i^{r'} = x_i^r \qquad \forall r = 0, ..., R-1, r' = r+1 \qquad (15)$$

By relaxing this set of constraints, if there is no other binding constraint in the set (13), the model forms a decomposible structure and the problem can be splitted into R subproblems. We used λ as the relaxation coefficient, which makes the objective in the extended model;

$$\min / \max \quad \mathcal{F}(x) + \sum_{r=1}^{R} \sum_{i=1}^{N} \lambda_i^r (x_i^0 - x_i^r) \qquad (1)$$

Decomposition for Simulation Calibration Problem

As mentioned before, the first set of computational experiments were done using the estimation error constraints (14.1) and (14.2) which are also binding constraints as they contain decision variables from two different dropout neural networks. As we did not want to relax them, the decomposition was also done such that each subproblem contained both the main neural network and one dropout neural network. One can easily obtain subproblems by taking $R = 2$ in the main model. The objective in each subproblem becomes;

$$\min \quad \sum_{j \in J} \frac{\delta_j^r}{(R-1)} + \sum_{i \in L_0} (\lambda_i^r - \lambda_i^{r-1}) x_i \tag{1.r}$$

Then, we utilized Lagrangian Decomposition method due to continuous input space. However, as the model size nearly doubled the decomposition did not present any computational benefits. The representation of reformulation and decomposition can be seen from Figure 5.11.

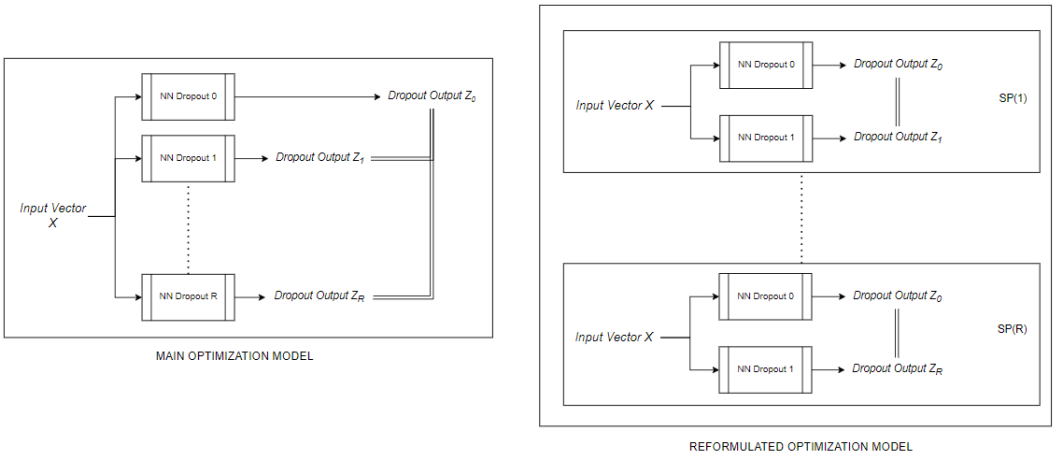

Figure 5.11: Trial Decomposition Approach Structure with Initial Uncertainty Constraints

Then, we decided on changing the estimation error constraints to allow better decomposition which are (14.1), (14.2) and (14.3). These constraints are used to control the maximum total difference between the target output and the dropout predicted outputs. Although they do not give the same results, a similar effect can also be obtained by minimizing differences for each dropout network separately which allows problem to be decomposed in R subproblems each having only one network calculation. So, the model constraints in each subproblem becomes

same as the base model introduced in the previous chapter. The objectives for subproblems can be written as;

$$\min \quad \sum_{j \in J} \gamma_j^r + \sum_{i \in L_0} (\lambda_i^r - \lambda_i^{r-1}) x_i \tag{1.r}$$

in which the objective of the problem is approximated as sum of the objectives subtract $\sum_{r=1}^{R} \sum_{j \in J} \gamma_j^r$. However, computational experiments shown similar results with no computational benefit of decomposition.

Decomposition for Assortment Optimization Problem

As mentioned in the other case study, due to (14.1) and (14.2) being binding constraints decomposition was done so that in each subproblem both the main network and one of the dropout networks remained as shown in Figure 5.11. So, the objective in each subproblem becomes;

$$\min \quad \frac{z^r}{(R-1)} + \sum_{i \in L_0} (\lambda_i^r - \lambda_i^{r-1}) x_i \tag{1.r}$$

Due to the input space being binary for this problem we decided on taking $\lambda = 0$ which completely separates the subproblems and using Branch&Bound Approach to achieve better computational efficiency. Hence, in each subproblem, the predicted revenue is maximized while approximated revenues from dropout networks are made sure to be in vicinity of the main prediction. Then, the assortments found in each subproblem is compared. The branching strategy was decided as; branch on product i which the assortment decision differs most across the subproblem solutions. As the parameter is binary, bounds are defined as; $x_i = 0$ and $x_i = 1$. The branching is continued until lower bound is equal to the upper bound. If any solution found such that assortments are same for all subproblems than the lower bound is updated. The detailed explanation can be seen in Algorithm 3.

To further improve the process, we also added a feasibility check at step 18 to find a solution to the problem. For each iteration, the solution x^{P_r} of the subproblem that gives the minimum objective value was tested to see if it also is a feasible solution to the main problem. This approach was motivated from

Algorithm 3 Branch and Bound for Assortment Problem

Input: Initial problem P with R subproblems
Output: Optimal assortment
Initialize the best found solution $x^* \leftarrow \emptyset$
Initialize the best found objective value $LB \leftarrow 0$
Initialize the upper bound $UB \leftarrow \infty$
Initialize a flag for convergence converged $\leftarrow$ false
while not converged **do**
    Initialize list X to store solutions and objective values for each subproblem
    for each subproblem P_r in P **do**
        Solve subproblem P_r to obtain solution x^{P_r} and objective value z^{P_r}
        Append (x^{P_r}, z^{P_r}) to $\mathcal{S}$
        if $\min z^{P_r} < UB$ **then**
            $UB \leftarrow \min z^{P_r}$
        end if
    end for
    Compute the variance for each product i across subproblems:

$$\bar{x}_i = \frac{1}{R}\sum_{r=1}^{R} x_i^r, \quad \sigma_i^2 = \frac{1}{R}\sum_{r=1}^{R}(x_i^r - \bar{x}_i)^2$$

    Identify the product i^* with the maximum variance:

$$i^* = \arg\max_i \sigma_i^2$$

    if all x^{P_r} are identical **then**
        $x^* \leftarrow x^{P_r}$
        $LB \leftarrow z^{P_r}$
    else
        for each subproblem P_r in P **do**
            Create subproblem P_{r0} by adding constraint $x_{i^*} = 0$ to P_r
            Create subproblem P_{r1} by adding constraint $x_{i^*} = 1$ to P_r
            Replace P_r in P with P_{r0} and P_{r1}
        end for
    end if
    if LB=UB **then**
        converged $\leftarrow$ true
    end if
end while
return x^*

the fact that all P_r objectives gives an upper bound to the P, which means the problem with minimum objective yields the best possible solution that can be found for P. The process can be represented as in Algorithm 4.

Algorithm 4 Feasibility Check for Subproblem Solutions

Input: Solution x^{P_r} and objective value z^{P_r} for subproblem P_r
Output: Updated lower bound LB if solution is feasible
if $P(x^{P_r})$ is feasible **then**
    if $z^{P_r} > LB$ **then**
        $LB \leftarrow z^{P_r}$
    end if
end if

From the computational experiments done with this algorithm, we found that for this problem the branching nearly always ended at root node or first iteration as the subproblem with minimum objective provided a feasible solution to the main problem. This results also indicated for this specific problem, optimizing the main problem with multiple dropouts gives same solution as optimizing the worst case scenario. Although we do not know which dropout network will give the minimum value, the problem can be adjusted to obtain best efficiency.

We then defined the measure of estimation error in the main problem by using the worst case scenario which is the minimum GSV value or predicted output z^r in this case. Then, the constraint set (14) can be written as;

$$z^r \geq \epsilon \qquad \forall r = 1, \ldots, R \tag{14}$$

which ensures all dropout outputs are greater than ϵ, hence, minimum output or worst case scenario is also greater than ϵ.

Notice that the model in this case have a decomposable structure as shown in Figure 5.10. The process of copying the input variables is done same as in the general case which makes the constraint (15), assortments should be same for each dropout, only binding constraint. After extending the model and decomposing it into subproblems, each subproblem constraint can be defined same as the base model from the previous chapter. Hence, we will have R subproblems that maximizes the predicted revenue. Then the objective of the main problem is found

by; $P(x^{P_r})|r = argmin_{r \in R} x^{P_r}$ with optimal assortment vector x^{P_r}. Similarly to first case study, computational experiments shown no significant computational benefit of decomposition.

5.5 Discussion

According to the results, we can say that dropout neural networks provide a good measure of estimation error for the output predictions. Various estimation error measures can be utilized within different problems by using dropout networks which further improve the reliability of optimal found from the suggested approach. Although we were not able to explore the computational benefits of decomposition in our case studies, by defining adequate constraints to control uncertainty that allows simpler decomposable structures and with parallelization of sub problems further improvements can be seen in the methodology.

Chapter 6

Conclusion and Future Search

There are various approaches introduced in literature to make causal inferences about the real-life scenarios. We studied how the input interventions should be altered to achieve some counterfactuals, and demonstrated how neural networks can be utilized within optimization models to achieve this. Our methodology, which combines machine learning with optimization techniques, has proven to be versatile and effective in handling problems with both integer and continuous parameter spaces across different domains.

Through the case studies presented-calibrating the FluTE simulation model and predicting revenues for a FMCG company with different assortments-we have shown that our approach can provide valuable insights into how changes in input parameters affect outcomes. We have also given some customized approaches to show how the methodology can be improved specific to the problem. Computational experiments in our base model show that the methodology, optimizing over neural networks, approximates the optimal parameter configurations relatively well while providing a big computational efficiency. The methodology can also be used together with other traditional approaches to balance the trade-off between the accuracy and the computational cost. This is also shown for one of our case studies by suggesting hybrid approaches.

We also suggest an approach to control estimation uncertainty in the approximated solutions by utilizing the dropout networks. The optimization is done while considering multiple approximations to improve our confidence interval. The methodology shows a great promise for providing estimation error bounds for the solutions. Computational results present multiple successful ways of employing the dropout networks with various measures of estimation error. We also provide an algorithm by decomposing the suggested methodology and utilizing branch and bound approach for problems with integer parameter space, binary in our case. Although the computational benefits for decomposition was not seen well in our case study as the original model already had a short runtime and the solutions were obtained easily, for more complex systems it may have significant benefits.

Future research could expand on this work by exploring the decomposition approaches for the model with estimation error bounds as it's structure is very open to it and it can provide great computational benefits. Especially for the problems with continuous spaces, although we only tried utilizing Lagrangian Decomposition method, our experiments were cut short due to model already having great computational efficiency similarly. While the suggested methodologies work well directly in our case studies, further improvements can also be made as we also showed with some customized approaches. Hence, for other specific problems the methodologies can be significantly improved by considering the suggested approaches even if we did not explore them deeply.

In conclusion, this study provides a foundation for continued exploration of machine learning-driven optimization models, with the potential to transform how we approach causal questions and make decisions based on hypothetical scenarios.

Bibliography

[1] N. Malleson, *Calibration of Simulation Models*, pp. 243–252. New York, NY: Springer New York, 2014.

[2] N. Pawlowski, D. Coelho de Castro, and B. Glocker, "Deep structural causal models for tractable counterfactual inference," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 857–869, Curran Associates, Inc., 2020.

[3] D. Fonseca, D. Navaresse, and G. Moynihan, "Simulation metamodeling through artificial neural networks," *Engineering Applications of Artificial Intelligence*, vol. 16, no. 3, pp. 177–183, 2003.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[5] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International Conference on Machine Learning (ICML)*, pp. 1058–1066, 2013.

[6] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 648–656, 2015.

[7] J. Ba and B. Frey, "Adaptive dropout for training deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 3084–3092, 2013.

[8] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pp. 1050–1059, 2016.

[9] Y. Gal, R. Islam, and Z. Ghahramani, "Deep bayesian active learning with image data," in *arXiv preprint arXiv:1703.02910*, 2017.

[10] D. Bertsimas and N. Kallus, "Predictive and prescriptive analytics for binary classification via a mixed-integer optimization approach," *Journal of Machine Learning Research*, vol. 21, no. 31, pp. 1–39, 2020.

[11] M. Sohoni, P. Sen, R. Aggarwal, and S. Kumar, "Surrogate modeling for aerodynamic optimization of aircraft configurations using deep neural networks," *AIAA Journal*, vol. 56, no. 11, pp. 4465–4476, 2018.

[12] A. Tamar, G. Thomas, H. Xu, D. Sarne, E. Brunskill, and B. Yonatan, "Sequential decision making with deep reinforcement learning and mixed integer programming," in *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[13] Y. Chen, Z. Wen, Y. Zhang, T. Davis, and D. Goldfarb, "Learning to optimize: Training deep neural networks for mixed-integer nonlinear programming," *INFORMS Journal on Computing*, vol. 31, no. 3, pp. 455–470, 2019.

[14] R. Anderson, J. Huchette, C. Tjandraatmadja, and J. P. Vielma, "Strong mixed-integer programming formulations for trained neural networks," *Mathematical Programming*, vol. 183, no. 1, pp. 3–39, 2020.

[15] G. Perakis and A. Tsiourvas, "Optimizing objective functions from trained relu neural networks via sampling," *arXiv preprint arXiv:2102.12829*, 2021.

[16] Z. Kang, J. A. Lozano, F. Caro, H. Ghaffari, and D. Bergman, "Optimizing over an ensemble of neural networks," *arXiv preprint arXiv:2107.00847*, 2021.

[17] G. Mustafaraj, D. Marini, A. Costa, and M. Keane, "Model calibration for building energy efficiency simulation," *Applied Energy*, vol. 130, p. 72–85, 10 2014.

[18] H. Lim and Z. J. Zhai, "Comprehensive evaluation of the influence of meta-models on bayesian calibration," *Energy and Buildings*, vol. 155, pp. 66–75, 2017.

[19] J. Chen, X. Gao, Y. Hu, *et al.*, "A meta-model-based optimization approach for fast and reliable calibration of building energy models," *Energy*, vol. 188, p. 116046, 2019.

[20] A. Patwary, W. Huang, and H. K. Lo, "Metamodel-based calibration of large-scale multimodal microscopic traffic simulation," *Transportation Research Part C: Emerging Technologies*, vol. 124, p. 102859, 2021.

[21] L. Van Gelder, P. Das, H. Janssen, *et al.*, "Comparative study of metamodelling techniques in building energy simulation: Guidelines for practitioners," *Simulation Modelling Practice and Theory*, vol. 49, pp. 245–257, 2014.

[22] F. Dunke and S. Nickel, "Neural networks for the metamodeling of simulation models with online decision making," *Simulation Modelling Practice and Theory*, vol. 99, p. 102016, 2020.

[23] A. M. Nezhad and H. Mahlooji, "An artificial neural network meta-model for constrained simulation optimization," *Journal of the Operational Research Society*, vol. 65, pp. 1232–1245, 2014.

[24] P. Rusmevichientong and H. Topaloglu, "Robust assortment optimization in revenue management under the multinomial logit choice model," *Operations Research*, vol. 58, no. 6, pp. 1666–1680, 2010.

[25] M. L. Fisher and K. Ramdas, *Handbook of Operations Analytics Using Data Envelopment Analysis*. Springer International Publishing, 2017.

[26] P. Bell and A. Morton, "Mixed integer programming in planning and scheduling: The state of the art," *European Journal of Operational Research*, vol. 152, no. 3, pp. 530–542, 2004.

[27] D. Bertsimas, J. Dunn, and S. Hu, "The price of fairness," *Operations Research*, vol. 66, no. 6, pp. 1599–1616, 2018.

[28] A. Belloni, R. Freund, and M. Selove, "Optimizing product assortments and prices: A nested logit model," *Management Science*, vol. 62, no. 8, pp. 2327–2345, 2016.

[29] A. G. Kök and M. L. Fisher, "Demand estimation and assortment optimization under substitution: Methodology and application," *Operations Research*, vol. 55, no. 6, pp. 1001–1021, 2007.

[30] G. Vulcano, G. van Ryzin, and R. Ratliff, "Estimating primary demand for substitutable products from sales transaction data," *Operations Research*, vol. 60, no. 2, pp. 313–334, 2012.

[31] Y. Huang, F. Xie, and J. Chen, "Intelligent assortment planning and optimization in online retailing using deep learning," *Electronic Commerce Research and Applications*, vol. 35, p. 100849, 2019.

[32] D. L. Chao, M. E. Halloran, V. J. Obenchain, and I. M. Longini, "Flute, a publicly available stochastic influenza epidemic simulation model," *PLoS Comput Biol*, vol. 6, no. 1, p. e1000656, 2010.

Appendix A

Calculation of Performance Metrics

A.1 R^2 Calculation

The formula for R^2 is given by:

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

where SS_{res} is the residual sum of squares and SS_{tot} is the total sum of squares. Define the following variables to calculate the SS_{res} and SS_{tot} ;

- y_i is the actual value of the output variable.
- $\hat{y}_i$ is the predicted value from the neural network model.
- $\bar{y}$ is the mean of the actual values.
- n is the number of data points.

Residual Sum of Squares ($SS_{\mathbf{res}}$)

The residual sum of squares is calculated as:

$$SS_{\text{res}} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Total Sum of Squares ($SS_{\mathbf{tot}}$)

The total sum of squares is calculated as:

$$SS_{\text{tot}} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

A.2 Mean Absolute Error (MAE) Calculation

The Mean Absolute Error (MAE) is a measure of the average value of errors in a set of predictions, without considering their direction. It is calculated as follows:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

A.3 Mean Absolute Relative Error (MARE) Calculation

The Mean Absolute Relative Error (MARE) is a measure of the average relative value of errors in a set of predictions. It is calculated as follows:

$$\text{MARE} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$