



SCHOOL of
GRADUATE STUDIES
EAST TENNESSEE STATE UNIVERSITY

East Tennessee State University
Digital Commons @ East Tennessee
State University

Electronic Theses and Dissertations

Student Works

8-2021

Applying Deep Learning to the Ice Cream Vendor Problem: An Extension of the Newsvendor Problem

Gaffar Solihu
East Tennessee State University

Follow this and additional works at: <https://dc.etsu.edu/etd>



Part of the [Applied Mathematics Commons](#), [Artificial Intelligence and Robotics Commons](#), [Data Science Commons](#), [Mathematics Commons](#), and the [Statistics and Probability Commons](#)

Recommended Citation

Solihu, Gaffar, "Applying Deep Learning to the Ice Cream Vendor Problem: An Extension of the Newsvendor Problem" (2021). *Electronic Theses and Dissertations*. Paper 3945. <https://dc.etsu.edu/etd/3945>

This Thesis - unrestricted is brought to you for free and open access by the Student Works at Digital Commons @ East Tennessee State University. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Digital Commons @ East Tennessee State University. For more information, please contact digilib@etsu.edu.

Applying Deep Learning to The Ice Cream Vendor Problem: An Extension of the
Newsvendor Problem

A thesis

presented to

the faculty of the Department of Mathematics & Statistics

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Science in Mathematical Sciences

by

Gaffar Olamide Solihu

August 2021

Jeff Knisley, Ph.D., Chair

Michele Joyner, Ph.D.

JeanMarie Hendrickson, Ph.D.

Keywords: machine learning, deep learning, simulation, supply chain, newsvendor,
inventory optimization, operations research.

ABSTRACT

Applying Deep Learning to The Ice Cream Vendor Problem: An Extension of the Newsvendor Problem

by

Gaffar Olamide Solihu

The Newsvendor problem is a classical supply chain problem used to develop strategies for inventory optimization. The goal of the newsvendor problem is to predict the optimal order quantity of a product to meet an uncertain demand in the future, given that the demand distribution itself is known. The Ice Cream Vendor Problem extends the classical newsvendor problem to an uncertain demand with unknown distribution, albeit a distribution that is known to depend on exogenous features. The goal is thus to estimate the order quantity that minimizes the total cost when demand does not follow any known statistical distribution. The problem is formulated as a mathematical programming problem and solved using a Deep Neural network approach. The feature-dependent demand data used to train and test the deep neural network is produced by a discrete event simulation based on actual daily temperature data, among other features.

Copyright 2021 by Gaffar Olamide Solihu

All Rights Reserved

ACKNOWLEDGMENTS

First and foremost, I want to express my gratitude to Dr. Jeff Knisley of the Department of Mathematics and Statistics at East Tennessee State University, who has constantly allowed me to grow through my thesis while guiding me in the proper path whenever I needed it.

I would also like to acknowledge my thesis committee members, Dr. Michele Joyner and Dr. JeanMarie Hendrickson of the Department of Mathematics and Statistics at East Tennessee State University. I am deeply indebted to them for their valuable comments on this thesis and the knowledge they have impacted on me throughout my years of study.

Finally, I want to convey my heartfelt gratitude to my parents and my lovely wife for their unwavering support and encouragement throughout my years of study, as well as during the research and writing of this thesis. This accomplishment would not have been possible without them.

TABLE OF CONTENTS

ABSTRACT	2
ACKNOWLEDGMENTS	4
LIST OF TABLES	7
LIST OF FIGURES	8
1 INTRODUCTION AND BACKGROUND	9
1.1 Supply Chain Management (SCM) and Inventory Optimization . . .	9
1.2 The Newsvendor Problem	10
1.3 Multi-Product Newsvendor Problem and It's Challenges	12
1.4 Application of Deep Learning to the Newsvendor (NV) Problem . .	15
1.5 Discrete Event Simulation with Simpy	19
2 STOCHASTIC LINEAR PROGRAMMING, MACHINE LEARNING AND	
DEEP LEARNING	21
2.1 Probability and Randomness	21
2.2 Stochastic Linear Programming	24
2.2.1 Linear Programming	24
2.2.2 Stochastic Linear Programming	25
2.3 Solutions to Stochastic Linear Programs	26
2.3.1 Approximation and Sampling Method	26
2.3.2 Data Science Machine Learning Approach	27
2.4 Machine Learning (ML)	28
2.4.1 Supervised Learning	28
2.4.2 Unsupervised Learning	33

2.4.3	Reinforcement Learning	34
2.5	Deep Learning	34
2.5.1	Hyperband Algorithm	39
3	THE ICE CREAM VENDOR PROBLEM	42
3.1	Derivation of the Ice Cream Vendor Problem	42
3.2	Simulation of Ice Cream Vendor	43
3.3	Linear Programming Approach	45
3.4	Deep Learning Approach	46
3.5	Implementation	47
3.6	Results and Conclusions	52
4	FUTURE DIRECTIONS	54
	BIBLIOGRAPHY	55
	APPENDICES	62
	Appendix A : Ice Cream Vendor Data Simulation	62
	Appendix B : Solutions to Ice Cream Vendor Problem	69
	B.1 Linear Programming Approach	69
	B.2 Deep Learning approach	71
	VITA	81

LIST OF TABLES

1	Brief Description of Notations	11
2	Simulated Demand Data	44
3	LP Predicted Order Quantity	46
4	DNN Training Log Table 1	51
5	DNN Training Log Table 2	51
6	DNN Test Data	53
7	DNN Predicted Order Quantity	53

LIST OF FIGURES

1	An illustration of a deep learning neural network [9].	16
2	A supervise ML process [3].	29
3	A linear regression model	31
4	An example of a decision tree	32
5	Examples of SVM	33
6	Structure of a DNN [29]	37
7	Leaky rectified linear activation function	49

1 INTRODUCTION AND BACKGROUND

The emergence of “Big data” has paved the way for sophisticated approaches for solving complicated problems which can best be likened to learning from documented history (data) to improve the future. The combination of big data and mathematical models is used to solve problems across industries like healthcare, education, banking and securities, communication, manufacturing, government, insurance, retail and wholesale trade, and many more [37, 55]. A particular example of such problems is that of inventory optimization.

In this thesis, we propose an approach based on deep learning to solve an inventory optimization problem called the Ice Cream Vendor Problem. This approach predicts the order quantities based on features of the demand data in order to satisfy uncertain demand and maximize expected profit.

1.1 Supply Chain Management (SCM) and Inventory Optimization

A supply chain can be defined as a network between a company and its suppliers to produce and distribute a specific product to the final buyer. It involves a series of steps to get a product or service to the final consumer [26]. Furthermore, the supply chain lays out all aspects of the production process, including the activities performed at each stage, information exchanged, natural resources converted into useful materials, human resources, and other components that go into the final product or service [11]. Supply chains can vary in complexity, with some having only a few stages while others have several. For example, a simple supply chain consists of a supplier, a manufacturer, and a retailer who sells to final consumers. However, the supply chains of large corporations often involve hundreds of facilities (retailers, distributors, plants, and suppliers) that are globally distributed and

involve thousands of parts and products.

In order to overcome the difficulties associated with complexities in a supply chain, the chain of supplies needs to be well managed [43]. With effective supply chain management in place, companies can cut excess costs and deliver products to the consumers faster [17].

For a highly effective supply chain management, it is essential that an inventory is kept and thoroughly optimized.

Inventory optimization is a collection of strategies designed to deliver the right amount of product at the right time for the lowest possible cost. That is, delivering the optimal balance between supply and demand. It includes the practice of having the right levels of inventory to meet target service levels while keeping a minimum amount of capital locked for inventory. The goal of an inventory optimization strategy is to maintain a steady flow of inventory, eliminate out-of-stock situations, mitigate loss and risk, all while boosting profits through improved efficiency and lower inventory costs[22].

In this thesis, we explore an extension of the newsvendor problem and its application to solving the inventory optimization problem.

1.2 The Newsvendor Problem

The newsvendor problem is a mathematical model in operations research used to determine optimal inventory decisions under uncertainties [34]. The classic newsvendor problem is a problem that is characterized by fixed prices and uncertain demand for a perishable good. As its name entails, a newspaper vendor buys x copies of a newspaper in the morning at a cost of $\$c$ per paper, he sells each for $\$p$ and expects a demand of D copies that day. The newsvendor's objective is to maximize his profit, or equivalently, to minimize his cost.

He must decide how many ordered x copies he will require to meet his objective, keeping in mind that surplus copies cannot be returned to his supplier, and that if he purchases fewer than the demand, he would disappoint his customers and lose money.

Table 1: Brief Description of Notations

Symbols	Description
c	cost price
p	sales price
D	uncertain demand
x	Order quantity
c_o	per-unit Overage cost
c_u	per-unit underage cost
S	Warehouse storage size
K	Capacity of the Truck
V_i	Volume of the goods
$[a]^+$	maximum of a and 0
E_D	Expected value in D

In general, the newsvendor problem assumes that a company purchases some goods at the beginning of a period and sells them during the period. At the end of the period unsold goods must be discarded, incurring a cost for overstocking (hold, salvage, dispose). In addition, if it runs out of goods in the middle of the period, it incurs a cost (shortage, back-order), losing potential profit [36]. These costs are called Underage cost (opportunity cost) and Overage cost respectively.

The newsvendor's cost function is thus

$$C(x; D) = \begin{cases} c_u(D - x) & \text{if } D > x \\ c_o(x - D) & \text{if } D < x \end{cases} \quad (1)$$

where c_u is per-unit underage cost and c_o is per-unit overage cost, see Table 1. If we use

the notation $[a]^+ = \max(a, 0)$, we can rewrite (1) as

$$C(x; D) = c_u[D - x]^+ + c_o[x - D]^+ \quad (2)$$

However, finding the order quantity x that maximizes the expected profit is equivalent to finding the order quantity x that minimizes the expected value of the sum of the underage and overage costs, given that demand D is a random variable [39]. That is, the optimal order quantity x_* can be obtained by solving the following optimization problem :

$$\min_x C(x; D) = \mathbf{E}_D \left(c_u[D - x]^+ + c_o[x - D]^+ \right) \quad (3)$$

As expressed in [36, 48], if $F(\cdot)$ is the cumulative density function of the demand distribution and F^{-1} is its inverse, then the optimal solution of (3) can be obtained as

$$x_* = F^{-1} \left(\frac{c_u}{c_u + c_o} \right) \quad (4)$$

where $c_u/(c_u + c_o)$ is called the critical fractile.

1.3 Multi-Product Newsvendor Problem and Its Challenges

In real-world problems ,companies rarely manage a single product, so it is important to extend the classic newsvendor problem to provide solutions for multiple products. The Multi-Product Newsvendor Problem (MPNP) is an extension of the classic newsvendor problem. At the beginning of a single period, a buyer is interested in determining order quantities x_i for i ($i = 1, 2, \dots, n$) products to satisfy the demand D_i , where the underage and overage cost is represented as

$$C_i(x_i; D_i) = \begin{cases} c_{iu}(D_i - x_i) & \text{if } D_i > x_i \\ c_{io}(x_i - D_i) & \text{if } D_i < x_i \end{cases} \quad (5)$$

We define the deterministic multi-product newsvendor problem as a linear program with constraints as

$$\begin{aligned}
\min_{x_i} \quad & \sum_{i=1}^n \left(c_i x_i + c_{ui} [D_i - x_i]^+ + c_{oi} [x_i - D]^+ \right) \\
\text{s.t.} \quad & \sum_{i=1}^n x_i \leq K \\
& \sum_{i=1}^n V_i x_i \leq S
\end{aligned} \tag{6}$$

Also, when the demand is stochastic (i.e., a random variable), with probability density function $f_i(\cdot)$ and cumulative distribution function $F_i(\cdot)$, the objective function becomes

$$\begin{aligned}
\min_x \quad & \mathbf{E}_D \left(\sum_{i=1}^n \left(c_i x_i + c_{ui} [D_i - x_i]^+ + c_{oi} [x_i - D]^+ \right) \right) \\
\text{s.t.} \quad & \sum_{i=1}^n x_i \leq K \\
& \sum_{i=1}^n V_i x_i \leq S
\end{aligned} \tag{7}$$

which is equivalent to


$$\begin{aligned}
\min_x \quad & \sum_{i=1}^n c_i x_i + \int_{-\infty}^{\infty} \left(c_{ui} [D_i - x_i]^+ + c_{oi} [x_i - D_i]^+ \right) f_i(D_i) dD_i \\
\text{s.t.} \quad & \sum_{i=1}^n x_i \leq K \\
& \sum_{i=1}^n V_i x_i \leq S
\end{aligned} \tag{8}$$

However, in the real world, the newsvendor problem is faced with some practical challenges. **Some of the challenges** are **estimating the probability distribution**, **estimating the underage and overage cost**, and also the **effect of some exogenous features**.

As mentioned in [63], early research mainly focused on refinement of distributional and mathematical method and solving the model as an optimization problem. For example, in

[27], they designed an algorithm for the price-dependent distribution method to include the influence of price. A method for the extension of an assumed distribution to multi-product cases was proposed by Zhou in [64]. In [2], they examined the newsvendor model in the presence of correlated demands. Specifically under a stationary Autoregressive model, the performance of a classic newsvendor solution vs. a dynamic forecast-based approach is investigated. Scarf in [42] first tried to solve the newsvendor problem with only sample mean \tilde{x} and sample variance $\hat{\sigma}^2$ and this approach was later expanded to multi-product case by calculating the demand for each item then adding them up [20].

Many approaches have concentrated on solving the problem by approximating the probability distribution and neglecting the effect of exogenous features (e.g., weather, day of the week, time of the day, location of business, etc.) on the demand. The multi-feature newsvendor problem (MFNP) extends the multi-product newsvendor problem to “Big Data” whose features can be used to address the challenges of the MPNP. In [40], they try to solve the multi-product newsvendor problem by proposing a machine learning algorithm to solve the problem. They claim that the algorithm can be extended to other situations, such as having a new item with limited data, censored demand data, ordering for multiple items and so on. They postulate that “the optimal base-stock level is related to the demand features via a linear function, that is $y^* = w^T x$, where x is the vector of features $\{(x^1, d^1), \dots, (x^n, d^n)\}$ and w is a vector of weights”.



In [40, 5], they estimate these weights by solving the optimization problem

$$\begin{aligned}
\min_w \quad & \frac{1}{n} \sum_{i=1}^n \left(c_u [d_i - w^T x_i]^+ + c_o [w^T x_i - d_i]^+ \right) + \lambda \|w\|_k^2 \\
\text{s.t.} \quad & [d_i - w^T x_i]^+ \geq d_i - w_1 - \sum_{j=2}^p w_i x_i^j; \forall i = 1, \dots, n. \\
& [w^T x_i - d_i]^+ \geq w_1 + \sum_{j=2}^p w_i x_i^j - d_i; \forall i = 1, \dots, n.
\end{aligned} \tag{9}$$

where n is the number of observations, p is the number of features, and $\lambda \|w\|_k^2$ is the regularization term. They claim that when the number of features is relatively small, the regularization term can be ignored. They also claim that this approach works better than other data-driven approaches such as sample average approximation (SAA) and separated estimation and optimization (SEO) if there is access to historical data.

Recently, advanced machine learning and neural network techniques, such as Support Vector Machine, recurrent neural networks, LSTM neural networks, etc. have been used to solve the MFNP as seen in [10, 46].

1.4 Application of Deep Learning to the Newsvendor (NV) Problem

Deep learning, or deep neural networks (DNN), is a branch of machine learning that uses multiple layers to define and train a function model. A deep neural network is an artificial neural network with multiple layers between the input and output as illustrated in Figure 1, that consist of neurons, synapses, weights, biases, and activations (nonlinear functions) [57]. Deep learning has been applied to various problems, such as speech recognition, image recognition, natural language processing, drug discovery, recommendation systems, demand prediction, and many more [57].

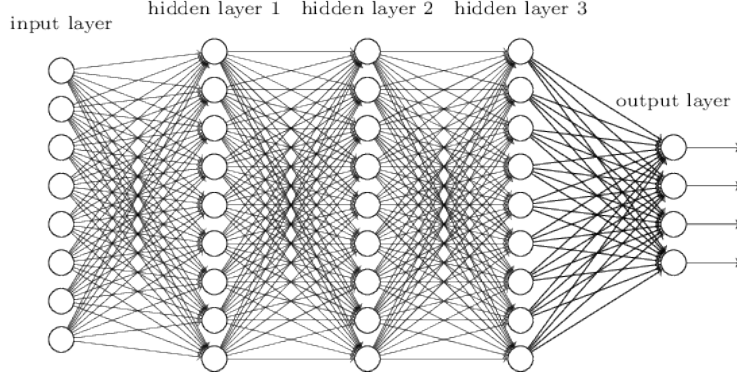


Figure 1: An illustration of a deep learning neural network [9].

To improve on the aforementioned machine learning algorithm approaches for solving the newsvendor problem shown in [40, 5], Oroojlooyjadid et al. in [36] extended on the machine learning methods applied in [5] by developing a new approach to solve the newsvendor problem. They proposed an algorithm based on deep learning that optimizes the order quantities for all products based on features of the demand data; this algorithm integrates the forecasting and inventory-optimization steps, rather than solving them separately, as it's typically done, and does not require knowledge of the probability distributions of the demand.

Referencing [36], Oroojlooyjadid et al. proposed a revised loss function of (9),

$$\min_w \sum_{i=1}^n l(\theta(x_i; w), y_i) + \lambda R(w) \quad (10)$$

where w is the matrix of the weights, x_i is the vector of the inputs, $\theta(\cdot)$ is the Deep Neural Network (DNN) activation function, and $R(w)$ is the regularization function with weight λ . The regularization term is typically l_1 or l_2 norm of the weights, which is used to prevent over-fitting. The revised loss function they proposed considers the impact of inventory shortage and holding costs and was aimed to measure the closeness of the outputs of the

model to the true values, which allows the deep learning algorithm to obtain the minimizer of the newsvendor cost function directly.

As illustrated in Figure 1, the DNN they used has a cascade of many layers of linear or nonlinear function. In each node $j(j = 1, \dots, n)$ of a layer $l(l = 1, \dots, n)$, the input value

$$z_j^l = \sum_{i=1}^n a_i^{l-1} w_{i,j} \quad (11)$$

is calculated and the value of the function $g_j^l(z_j^l)$ provides the output value of the node, where the function $g_j^l(\cdot)$ is called the activation function and the value of $g_j^l(z_j^l)$ is called the activation of the node denoted as a_j^l . The data is represented as

$$[(x_i^1, d_i^1), \dots, (x_i^m, d_i^m)]_{i=1}^n$$

where $x_i^q \in \mathbb{R}^p$ and $d_i^q \in \mathbb{R}$ for $i = 1, \dots, n$ and $q = 1, \dots, m$. The mathematical problem is

$$E_i = \min_{y_i^1, \dots, y_i^m} \frac{1}{m} \left(\sum_{q=1}^m c_u [d_i^q - y_i^q]^+ + c_o [y_i^q - d_i^q]^+ \right) \quad (12)$$

where E_i is the loss value of period i and $E = \frac{1}{n} \sum_{i=1}^n E_i$ is the total loss. Since one of the two terms must be zero, the loss function (12) can be written as

$$E_i^q = \begin{cases} c_u (d_i^q - y_i^q) & \text{if } y_i^q < d_i^q \\ c_o (y_i^q - d_i^q) & \text{if } d_i^q \leq y_i^q \end{cases} \quad (13)$$

They also proposed a revised Euclidean loss function, which is the square of (12) that penalizes the order quantities that are far from d_i , much more than those that are close, and claimed that sometimes leads to a better solution and the gradient is available in the whole solution space, as follows:

$$E_i = \min_{y_i^1, \dots, y_i^m} \frac{1}{m} \left(\sum_{q=1}^m \left(c_u [d_i^q - y_i^q]^+ + c_o [y_i^q - d_i^q]^+ \right)^2 \right) \quad (14)$$

Then they have

$$E_i^q = \begin{cases} \frac{1}{2} \|c_u(d_i^q - y_i^q)\|_2^2 & \text{if } y_i^q < d_i^q \\ \frac{1}{2} \|c_0(y_i^q - d_i^q)\|_2^2 & \text{if } d_i^q \leq y_i^q \end{cases} \quad (15)$$

The gradients and sub-gradient under the loss functions (13) and (15) respectively are

$$\frac{\partial E_i^q}{\partial w_{jk}} = \begin{cases} a_j^l \delta_j^l(u) & \text{if } y_i^q < d_i^q \\ a_j^l \delta_j^l(o) & \text{if } d_i^q \leq y_i^q \end{cases} \quad (16)$$

$$\frac{\partial E_i^q}{\partial w_{jk}} = \begin{cases} (d_i^q - y_i^q) a_j^l \delta_j^l(u) & \text{if } y_i^q < d_i^q \\ (y_i^q - d_i^q) a_j^l \delta_j^l(o) & \text{if } d_i^q \leq y_i^q \end{cases} \quad (17)$$

where $a_j^l = g_j^l(z_j^l)$, $\delta_j^l(u) = c_u(g_j^l)'(z_j^l)$ and $\delta_j^l(o) = c_o(g_j^l)'(z_j^l)$.

The gradients are used to iteratively update the weights of the network. **Stochastic Gradient Descent (SDG) algorithm with a fixed momentum of 0.9** is used to obtain new weights. **Oroojlooyjadid et al. ended up with two DNN models called DNN- l_1 from the Linear loss function (13) and DNN- l_2 from the Euclidean loss function (15) respectively.**

They generated 100 fully connected networks with random structures, where in each of the networks, the number of hidden layers is randomly selected as either two or three, the number of nodes in each hidden layer is also selected randomly based on the number of nodes in the layer before it, and also the learning rate and regularization parameters are drawn uniformly from $[10^{-5}, 10^{-2}]$. Ultimately, in order to select the best model, they use the **Hyperband** algorithm [28], **by which they trained each of the 100 networks for one epoch, then obtained the result on the test set, and removed the worst 10% of the network. Then run another epoch on the remaining networks and remove the worst 10%, then repeats the process to obtain the best model.** The Tensorflow library was used to implement this

procedure. Google created and published the Tensorflow library, a python-based framework for rapid numerical processing [1].

Finally, in order to check the validity of their algorithm, they implemented their DNN algorithms, Empirical Quantile (EQ) model in [7], the Linear Machine Learning (LML) and Kernel Regression (KR) in [40], the K-Nearest Neighbor (KNN) and Random Forest (RF) in [6] and the Separated Estimation Optimization (SEO) in [53]. They tested the algorithms on a real-world retailer’s basket dataset and multiple simulated datasets and concluded that their algorithm is superior to others when data is noisy and can also get close to optimal solutions when the data are noise-free. The code used to implement the DNN is open to public and can be found on Oroojlooyjadid’s repository on GitHub (<https://github.com/oroojlooy/newsvendor>).

In chapter 3, we discuss how we applied a variation of their DNN algorithms to solve the Ice cream Vendor problem by adapting a Pytorch version of DNNs discussed above from Karn Watcharasupat’s repository on GitHub.

1.5 Discrete Event Simulation with Simpy

The goal of newsvendor problems is to find optimal strategies for inventory optimization, but these strategies cannot be verified as optimal using real world data. Instead, optimal strategies tend to be developed via data produced by controlled simulations, after which such strategies are applied to actual data [61].

Discrete event simulation (DES) models a real world system that can be decomposed into a set of logically separate processes that produce discrete events over time. It is useful for evaluating different circumstances in a system without expensive field experiments, such

as simulating the day-to-day operation of a warehouse under different circumstances.

SimPy is an open-source package written in Python that is used for discrete-event simulation (DES). According to [31], “processes in SimPy are defined by Python generator functions and may, for example, be used to model active components like customers, vehicles or agents”. It describes that in the operation of a warehouse, whereby the purchase orders reduces the inventory and inventory been replenished time to time, a typical Simpy variable would be the inventory.

We considered several exogenous features and produced demand data that are dependent on these features such as temperature, precipitation, wind speed, relative humidity and so on.

Our goal is to collect feature driven Big Data for uncertain demand from a discrete events simulation, which represents real-world event, that will be used for data driven solution to the Newsvendor problem.

2 STOCHASTIC LINEAR PROGRAMMING, MACHINE LEARNING AND DEEP LEARNING

In order to understand the role of uncertainty in newsvendor problems, we introduce the concept of probability and its relationship to stochastic linear programming.

2.1 Probability and Randomness

Probability is the chance (likelihood) of an event happening. It is a mathematical description of randomness and uncertainty. Some relevant definitions and properties of probability will be introduced in order to understand stochastic linear programming.

A random experiment is a mechanism by which we observe uncertain outcome. The set of all possible outcomes (result of random experiment) is called the sample space \mathcal{S} . An event is a subset of the sample space [23].

Definition 2.1 (Event Space [23]) *The collection \mathcal{F} of subsets of the sample space \mathcal{S} is called an event space if*

\mathcal{F} is non-empty,

if $\mathcal{A} \in \mathcal{F}$, then $\mathcal{S} \setminus \mathcal{A} \in \mathcal{F}$,

if $\mathcal{A}_1, \mathcal{A}_2, \dots \in \mathcal{F}$ then $\bigcup_{i=1}^{\infty} \mathcal{A}_i \in \mathcal{F}$.

Definition 2.2 (Probability Space [23]) *A probability space is a triple $(\mathcal{S}, \mathcal{F}, \mathbb{P})$ of objects such that*

(a) \mathcal{S} is a non-empty set,

(b) \mathcal{F} is an event space of subsets of \mathcal{S} ,

(c) \mathbb{P} is a probability measure of $(\mathcal{S}, \mathcal{F})$

Definition 2.3 (Conditional Probability [23]) If $\mathcal{A}, \mathcal{B} \in \mathcal{F}$ and $\mathbb{P}(\mathcal{B}) > 0$, the conditional probability of \mathcal{A} given \mathcal{B} is denoted by $\mathbb{P}(\mathcal{A} \mid \mathcal{B})$ and defined by

$$\mathbb{P}(\mathcal{A} \mid \mathcal{B}) = \frac{\mathbb{P}(\mathcal{A} \cap \mathcal{B})}{\mathbb{P}(\mathcal{B})}$$

Definition 2.4 Events \mathcal{A} and \mathcal{B} of a probability space $(\mathcal{S}, \mathcal{F}, \mathbb{P})$ are called independent if

$$\mathbb{P}(\mathcal{A} \cap \mathcal{B}) = \mathbb{P}(\mathcal{A})\mathbb{P}(\mathcal{B}),$$

and dependent otherwise [23].

A random variable X is a function from the sample space \mathcal{S} to the real numbers, that is, $X : \mathcal{S} \rightarrow \mathbb{R}$ [23]. A random variable can either be discrete or continuous. A random variable X is said to be discrete, if its possible values (range) form a countable set and its distribution can be described by a probability mass function (pmf).

Definition 2.5 (Probability Mass Function) The probability mass function (or pmf) of a discrete random variable X is the function $p_X(x) : \mathbb{R} \rightarrow [0, 1]$ defined by

$$p_X(x) = \mathbb{P}(X = x)$$

Thus, $p_X(x)$ is the probability that the mapping X takes value x [23].

Definition 2.6 (Expected Value) If X is a discrete random variable, the expectation of X is denoted by $\mathbb{E}(X)$ and defined by

$$\mathbb{E}(X) = \sum_x x\mathbb{P}(X = x) = \sum_x xp_X(x)$$

The expectation of \mathbf{X} is often called the expected value or mean of \mathbf{X} [23].

Definition 2.7 *If \mathbf{X} is a random variable on $(\mathcal{S}, \mathcal{F}, \mathbb{P})$, the cumulative distribution function of \mathbf{X} is the function $\mathbf{F}_X : \mathbb{R} \rightarrow [0, 1]$ defined by*

$$\mathbf{F}_X(x) = \mathbb{P}(\mathbf{X} \leq x).$$

A random variable X is said to be continuous, if it takes infinite number of possible values. A continuous random variable is defined over an interval of values, and is represented by the area under a curve (integral).

Definition 2.8 *A random variable \mathbf{X} is continuous if its cumulative distribution function \mathbf{F}_X may be written in the form*

$$\mathbf{F}_X(x) = \mathbb{P}(\mathbf{X} \leq x) = \int_{-\infty}^x f_X(u) du \quad \text{for } x \in \mathbb{R}$$

for some non-negative function f_X . In this case, we say \mathbf{X} has a probability density function (or pdf) f_X [23].

Definition 2.9 *If X is a continuous random variable with density function f_X , the expectation of \mathbf{X} is denoted by $\mathbb{E}(\mathbb{X})$ and defined by*

$$\mathbb{E}(\mathbb{X}) = \int_{-\infty}^{\infty} x f_X(x) dx$$

Also, the total area under the probability density function is always equal to 1, that is,

$$\int_{-\infty}^{+\infty} f_X(x) dx = 1$$

2.2 Stochastic Linear Programming

2.2.1 Linear Programming

Linear programming is the optimization of a linear function, called the objective function, which is subject to a set of linear constraints. The simplicity of linear functions makes linear models easy to formulate, interpret, and analyze. Linear programs are particularly important because they accurately represent many practical applications of optimization [19].

A linear programming problem can be described as follows:

$$\begin{aligned} & \text{minimize} && z = c_1x_1 + \cdots + c_nx_n \\ & && a_{11}x_1 + \cdots + a_{1n}x_n = b_1 \\ & \text{subject to} && \vdots \\ & && a_{m1}x_1 + \cdots + a_{mn}x_n = b_m \\ & && x_j \geq 0, j = 1, \dots, n \end{aligned} \tag{18}$$

For $i = 1, \dots, m$. and $j = 1, \dots, n$., the x_j are non-negative decision variables, c_j are coefficients associated with the decision variables, a_{ij} and b_i represent the constant associated with the constraints. Let $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$, $\mathbf{c} = (c_1, c_2, \dots, c_n)^T$, $\mathbf{b} = (b_1, b_2, \dots, b_m)^T$, and \mathbf{A} = matrix of (a_{ij}) , the linear programming problem above can be represented as follows:

$$\begin{aligned} & \text{minimize} && \mathbf{c}^T \mathbf{x} \\ & \text{subject to} && \mathbf{Ax} = \mathbf{b} \\ & && \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{19}$$

Linear programming (LP) problems can be classified into deterministic and stochastic optimization problems base on the type of parameters (data) involve. We say a linear programming problem is deterministic if the data associated with the problem are known accurately, while a LP problem is stochastic if the data has elements of uncertainty or

randomness, such as future price of a certain product, product demand during a specific period, and many more.

2.2.2 Stochastic Linear Programming

Stochastic linear programming problems represent real-world problems because uncertainty is embedded into the model and it takes the advantage that the probability distribution of the data is known or can be estimated. In stochastic linear programming, the parameters become random variables [52].

The concept of recourse, or the ability to take remedial action after a random event, lies at the heart of stochastic linear programming. We minimize the cost of the first-period decision in addition to the expected cost of the second-period recourse decision as follows:

$$\begin{aligned} & \text{minimize} && \mathbf{c}^T \mathbf{x} + \mathbf{E}_w \mathbf{Q}(\mathbf{x}, \mathbf{w}) \\ & \text{subject to} && \mathbf{Ax} = \mathbf{b} \\ & && \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{20}$$

where $\mathbf{E}_w \mathbf{Q}(\mathbf{x}, \mathbf{w})$ is the optimum value of the second stage problem and w is a random event, then $\mathbf{Q}(\mathbf{x}, \mathbf{w})$ is

$$\begin{aligned} & \text{minimize} && \mathbf{d}(\mathbf{w})^T \mathbf{y} \\ & \text{subject to} && \mathbf{T}(\mathbf{w})\mathbf{x} + \mathbf{W}(\mathbf{x})\mathbf{y} = \mathbf{h}(\mathbf{w}) \\ & && \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{21}$$

The second linear program minimizes the cost $\mathbf{d}^T \mathbf{y}$ and describes how to choose $y(w)$, subject to some recourse constraints. This is also known as two-stage stochastic programming [44].

The Newsvendor problem is an example of a stochastic linear programming problem, because the vendor must decide now how much order quantity is needed to meet future unpredictable (stochastic) demand while minimizing cost (i.e. maximizing profit) under certain

limitations.

2.3 Solutions to Stochastic Linear Programs

The main objective of solving stochastic linear program is to find the optimal value for model parameters influenced by random event. The primary idea behind stochastic programming is to turn a stochastic problem into an appropriate deterministic problem that can be addressed with a suitable classical or modern numerical technique. In this thesis, we discuss the sampling method and the data science machine learning method for solving stochastic linear problems.

2.3.1 Approximation and Sampling Method

The sample average approximation (SAA) method is a Monte Carlo simulation-based approach to solving stochastic optimization problems. A sample average estimate derived from a random sample is often used to approximate the expected objective function of the stochastic problem, that is the recourse function $Q(x, w)$ in (12) is replaced with a Monte-Carlo estimate

$$\mathbf{E}_w \mathbf{Q}(\mathbf{x}, \mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \mathbf{Q}(\mathbf{x}, \mathbf{w}_k)$$

and (12) becomes as expressed in [32]

$$\begin{aligned} & \text{minimize} && \mathbf{c}^T \mathbf{x} + \frac{1}{N} \sum_{i=1}^N \mathbf{Q}(\mathbf{x}, \mathbf{w}_k) \\ & \text{subject to} && \mathbf{Ax} = \mathbf{b} \\ & && \mathbf{T}(\mathbf{w})\mathbf{x} + \mathbf{W}(\mathbf{x})\mathbf{y} = \mathbf{h}(\mathbf{w}) \\ & && \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0} \\ & && k = 1, 2, \dots, N. \end{aligned} \tag{22}$$

Various applications of stochastic optimization have effectively employed sampling-based approaches, such as, asset liability management in [24], supply chain network design in [41],

and many more.

2.3.2 Data Science Machine Learning Approach

The newsvendor model is an example of a stochastic linear programming problem , where we have the problem of minimizing the cost function in order to estimate the optimum order quantity X , that is

$$\min_x \frac{1}{n} \sum_{i=1}^n \left(c_u [D_i - X]^+ + c_o [X - D_i]^+ \right) \quad (23)$$

possibly subject to linear constraints.

This problem becomes complex when there is large demand data, where order quantity becomes a function of some features z_i and parameters β , that is, $X = z_i^T \beta$, and the problem becomes

$$\min_{\beta} \frac{1}{n} \sum_{i=1}^n \left(c_u [D_i - z_i^T \beta]^+ + c_o [z_i^T \beta - D_i]^+ \right) \quad (24)$$

possibly subject to linear constraints.

Its equivalent linear program, if $s_i = D - z_i^T \beta$ and $t_i = z_i^T \beta - D$ becomes

$$\begin{aligned} & \underset{\beta}{\text{minimize}} && \frac{1}{n} \sum_{i=1}^n (c_u s_i + c_o t_i) \\ & \text{subject to} && s_i \geq D - z_i^T \beta \\ & && t_i \geq z_i^T \beta - D \\ & && s_i \geq 0, t_i \geq 0 \end{aligned} \quad (25)$$

the machine learning model solves the linear program and produce optimum parameters β^* , which is then used to estimate order quantity given new datasets, that is

$$X^* = z_{new}^T \beta^*$$

To solve multistage stochastic programming problem, a machine learning technique was used in [12].

2.4 Machine Learning (ML)

Machine learning is the process of creating algorithms (systems) that extract useful information from data automatically [13]. Machine learning is basically the concept of teaching a computer program or algorithm how to improve over time at a given task.

Data, mathematical model, and ‘learning’ are the three concepts at the core of a machine learning algorithm. There are basically three classes of machine learning algorithms: supervised learning, unsupervised learning, and reinforcement learning.

2.4.1 Supervised Learning

Supervised learning is a type of machine learning that makes use of labeled datasets. These datasets are divided into train and test sets and used to train or “supervise” algorithms so that they can properly categorize data or predict outcomes [14]. The train dataset has output variable which needs to be predicted or classified. These algorithms can be classified into Regression and Classification algorithms. Simple Linear Regression, Decision Tree Regression, Support Vector Regression are examples of regression algorithms, while K-Nearest Neighbours, Decision Tree, Support Vector Machine are examples of classification algorithms. Also, some of these algorithms can be leveraged for both classes [3]. Figure 2 illustrates a typical supervised machine learning process. Some of the applications of Supervised learning models are spam detection, sentiment analysis, weather forecasting, and pricing predictions [14]. We’ll go through some of the most common supervised machine learning algorithms and discuss how they use data to learn and generate predictions.

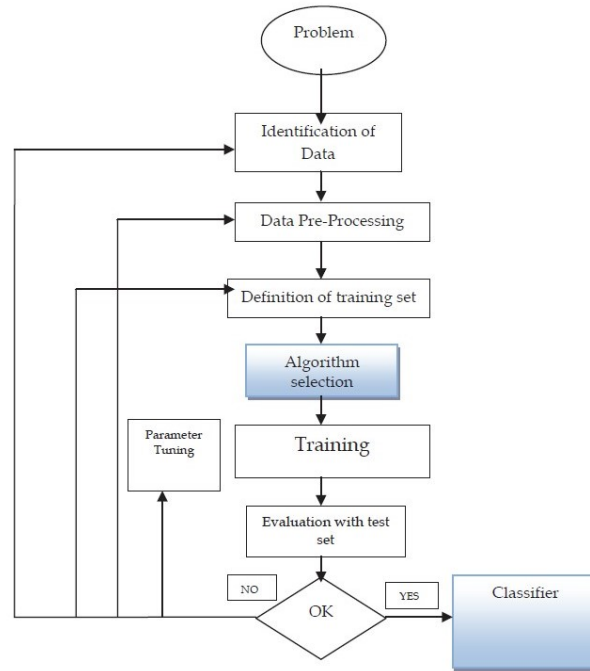


Figure 2: A supervise ML process [3].

k-Nearest Neighbors (KNN) : It is arguably the simplest machine learning algorithm, which assumes that similar observations exist in close proximity, and the algorithm makes predictions based on distance between points. In it's simplest form it considers one nearest neighbor which is closest to the training data point of interest and it's prediction is simply identifying such data point. In the case of large datasets, instead of considering only one closest neighbor, we can also consider k arbitrary number of neighbors [33]. There are several ways of calculating proximity and the choice of calculation depends on the problem, but the most popular choice is the euclidean distance (straight line distance). In practice KNN algorithm depends entirely on two questions , which are, how to calculate distance and what number of k will be efficient. The choice of k can simply be determined by running the algorithm several times with different values of k. The best k reduces the error on new

dataset. It can also be used for both classification and regression problems. See [38] for implementation in Python.

Linear Regression : Linear regression, also known as ordinary least square, is the most classic method for regression which establishes the relationship between independent and dependent variables by fitting a best line which is known as regression line [33]. The linear model can be represented in its simplest form as an equation of a straight line

$$\hat{Y} = b_0X + b_1$$

where \hat{Y} is the dependent variables (target), X is the independent variables (features), b_0 is the slope and b_1 is the intercept. The model finds the parameters b_0 and b_1 that minimize the mean square error between predictions and actual targets on the training dataset as seen in Figure 4. There are mainly two types, Simple (one independent variable) and Multiple Linear Regression is used for training set with more than one independent variable. When model becomes complex, where the parameters predicts well but are large, then this problem can be solved by introducing the regularization which then extends the model to either Ridge regression or LASSO, see [58]. Examples and implementation can be seen in [38].

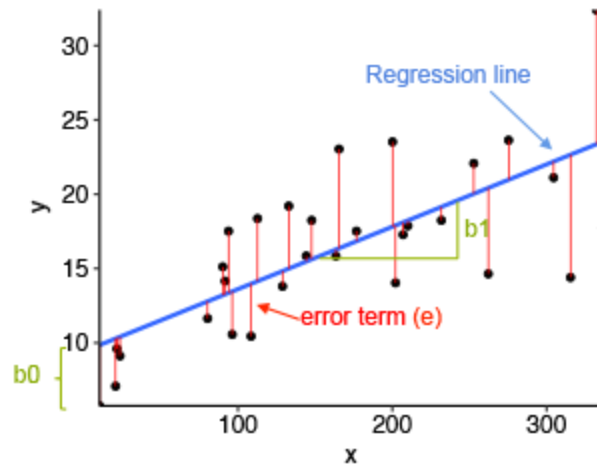


Figure 3: A linear regression model

Decision Tree : Decision trees are widely used machine learning algorithms for both classification and regression problems [33]. It is a decision support model that learns a hierarchy of if/else questions that lead to a decision. It consists of the root nodes, branches and the terminal nodes and uses different types of algorithm such as Gini, Chi-Square, Information Gain, and so on, to decide on how it splits into branches. Scikit-Learn uses the Classification and Regression Tree (CART) to train Decision trees, by splitting the training set into two subsets using a single feature and threshold. Once successful, it splits the subset using the same approach, then continues the process until it reaches the maximum depth defined or cannot find a split that will reduce impurity [33]. It is a very useful tool in data mining and it is easy to understand even with complex data. See [38] for implementation of Decision Tree in Python.

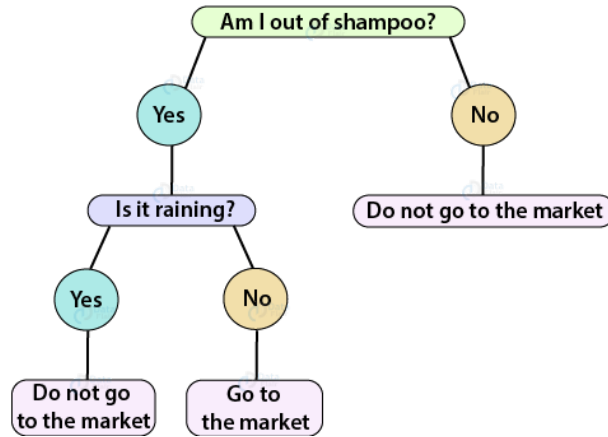


Figure 4: An example of a decision tree

Support Vector Machine (SVM) : SVM is one of the most widely used machine learning technique for classification problems. It works on the principle of margin calculation. In its simplest form, it draws margin in such a way that the distance between the margin and the classes is maximized, hence reducing classification error. It constructs a hyperplane (a flat affine subspace of dimension $N-1$, i.e. a line in 2D or a flat plane in 3D) or set of hyperplanes in a high-dimensional space, which can be used for classification, regression, or outlier detection. A good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class [62]. The classifier can be linear or nonlinear. If the classifier is linear, the algorithm depends on the subset of the training data points, typically the ones that lie on the border between the classes. These data points are called the support vectors and they are equidistant from the hyperplane as seen in Figure 5a. In real-world problems, many datasets are not linearly separable. One approach to handling nonlinear datasets is to add more features such as polynomial features. If the polynomial degree is low, it won't be able to handle complex datasets.

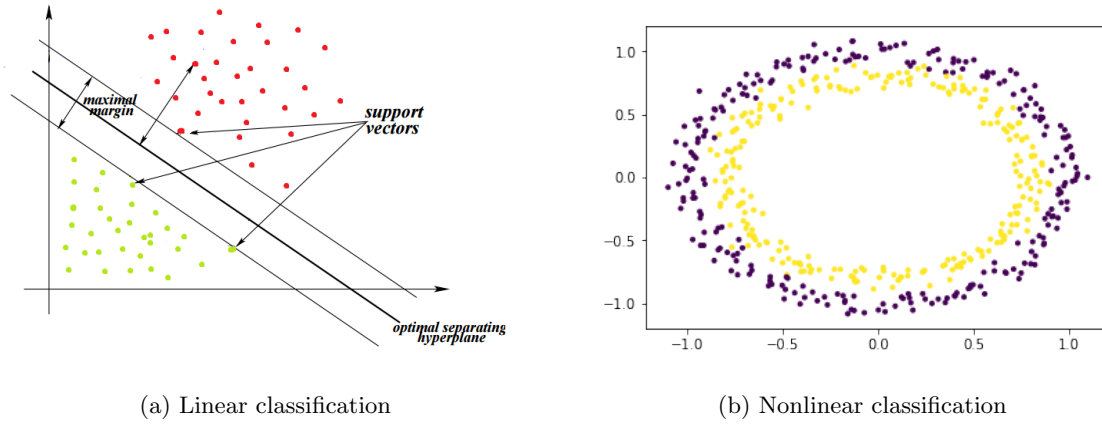


Figure 5: Examples of SVM

Also, with a high polynomial degree it creates a huge number of features making the model too slow. This problems can be addressed using the kernel trick [33]. Scikit-learn has a comprehensive introduction and implementation of SVM maching learning algorithm, see [38].

2.4.2 Unsupervised Learning

Unsupervised machine learning uses machine learning algorithms to analyze unlabeled data [14]. It learns features (patterns) from the data and when new data is introduced, it recognizes the data's class using previously learned features without the need for human intervention. These algorithms are mainly used for data clustering, association and feature reduction [15]. Example are Principal Component Analysis (PCA), Singular Value Decomposition (SVD), K-means Clustering, and so on. These models are applied in anomaly detection, recommendation engines, medical imaging, and many more.

Principal Component Analysis (PCA) : PCA is the most popular dimensionality

reduction algorithm. It is a process that converts observations of possibly correlated variables into linearly uncorrelated variables through orthogonal transformation [30]. It can be thought of as a projection method where data with m columns is projected into a lower-dimensional data while preserving as much variance as possible. It uses the Singular Value Decomposition (SVD) to calculate the eigenvalues and eigenvectors of the covariance matrix of the dataset. The eigenvectors represent the principal components and the eigenvalues represents the magnitudes for the components.

2.4.3 Reinforcement Learning

Reinforcement learning is a type of machine learning in which the learner makes judgments about which activities to do in order to improve the outcome. Until a situation is presented, the learner has no idea what steps to take and its actions now may have an impact on future situations [15]. It solely depends on trial and error search and delayed outcome [51]. Reinforcement learning algorithms interact with an environment, so there is a feedback loop between the learning system and its experiences.

As explained in [25], the model consists of a discrete set of environment states, a discrete set of agents actions, and a set of scalar reinforcement signals, typically $\{0,1\}$. These models are commonly applied in industry automation, self-driving cars, natural language processing, healthcare, trading automation and many more [29].

2.5 Deep Learning

Deep learning, also known as Deep Neural Networks, is a machine learning approach that learns several layers of data representations or features and generates cutting-edge

prediction results [21]. It is an artificial neural network with multiple layers between inputs (data) and outputs [36]. Deep neural networks have the capacity to model complex nonlinear relationships and it is suitable for dealing with complex large datasets [45].

Figure (1) depicts a deep neural network with 1 input layer, 3 hidden layers, and 1 output layer, that are connected. Weight averaging and activation function are core attributes of the connectivity of a deep neural network.

In this thesis, we applied deep learning approach to solve an extension of a newsvendor problem. The following are definitions and theorems relevant in understanding deep learning process.

Definition 2.10 (Convex Set [8]) A set Ω is **convex** if the line segment between any two points in Ω lies in Ω , that is $\forall x_1, x_2 \in \Omega$ and $\theta \in [0, 1]$

$$\theta x_1 + (1 - \theta)x_2 \in \Omega$$

In general, a convex combination of points $x_1, x_2, \dots, x_k \in \Omega$ is any point of form $\theta_1 x_1 + \theta_2 x_2 + \dots + \theta_k x_k$, where $\theta_i \geq 0$, $i = 1, 2, \dots, k$, and $\sum_{i=1}^k \theta_i = 1$.

Definition 2.11 (Convex function) A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, is convex if its domain is a convex set and $\forall x_1, x_2$ in its domain and all $\theta \in [0, 1]$, we have that

$$f(\theta x_1 + (1 - \theta)x_2) \leq \theta f(x_1) + (1 - \theta)f(x_2)$$

Thus, a function f is convex if and only if its epigraph, the set of all points above the function graph, is a convex set. A function is strictly convex if $\forall x_1, x_2$ in its domain and all $\theta \in [0, 1]$, we have that

$$f(\theta x_1 + (1 - \theta)x_2) < \theta f(x_1) + (1 - \theta)f(x_2)$$

Theorem 2.1 *If $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is convex and $g : \mathbb{R}^m \rightarrow \mathbb{R}^l$ is convex, then*

$$g \circ f : \mathbb{R}^n \rightarrow \mathbb{R}^l$$

is convex.

Theorem 2.2 *If a function f is convex on a compact set Ω , then f has a global minimum on Ω .*

Artificial neural networks (ANNs), often known as neural networks are comprised of node layers, containing an input layer, hidden layer(s), and an output layer. Each node is connected to the others and has a weight and threshold assigned to it. If a node's output exceeds a certain threshold value, the node is activated, and data is sent to the next layer of the network [16]. Neural networks are core to deep learning algorithms.

The most significant distinction between neural networks and linear models is that a neural network's nonlinearity causes most loss functions to become non-convex. This means that neural networks are usually trained by using iterative, gradient-based optimizers that do not guarantee convergence [21]. Deep learning algorithms can process unstructured data, like images and text. Then through the processes of gradient descent and backpropagation adjusts and fits itself for accuracy, allowing it to make predictions about a new data accurately. These algorithms are possible today due to the emergence of faster computers with larger memory and the availability of larger datasets which are used to train the networks.

Figure (6) depicts the structure of a DNN and illustrates how it works. Let \mathbf{X} be the input matrix of data points and W_{ih} be the weights matrix to the hidden layer with some biases \mathbf{b}_h . The first layer can be model as a function (linear transformation) that takes in \mathbf{X} and outputs \mathbf{Z}_1 . Then \mathbf{Z}_1 is passed into an activation function $\sigma(\cdot)$ in the second

layer, which can be modelled as a function with input \mathbf{h}_1 and outputs \mathbf{Z}_2 . \mathbf{Z}_2 is then passed into another activation function, this process can be understood mathematically as the composition of functions.

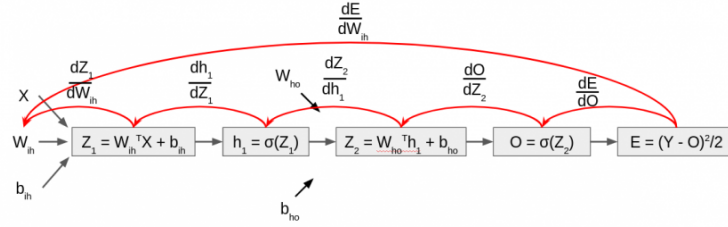


Figure 6: Structure of a DNN [29]

The forward movement is known as forward propagation and at each layer an activation function is applied so as to get a desirable output for the next layer and the same process is repeated. Subsequently, error is calculated from the output and a back-propagation process (partial differentiation of functions) updates the weights and the biases. To compute an error a loss function is employed.

A loss function is a function used to assess how effectively an algorithm models a dataset [59]. Typically, if the prediction is far from the actual, the loss value will be high, otherwise, the loss function outputs a lower value for a pretty good prediction. They are also sometimes referred to as cost function especially when coupled with regularization parameters. Loss functions can be classified into Regression loss functions and Classification loss functions. The choice of loss function depends on the problem we are trying to solve. Some common examples are

Mean Square Error (MSE) : which averages the squared difference between the actual value

and the predicted value.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_{actual} - y_{predicted})^2$$

Mean Absolute Error(MAE) : which measure as the average of sum of absolute differences between predicted value and actual value

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_{actual} - y_{predicted}|$$

Cross Entropy Loss(CEL) : commonly used for classification problems, it increases as the predicted probability diverges from the actual label.

$$CEL = - (y_i \log(y_i) + (1 - y_i) \log(1 - y_i))$$

Activation functions are functions that are employed in neural networks to compute the weighted sum of input and biases, as well as impart non-linearity in the network's hidden layers, which separates neural network algorithms from linear regression models. It can be represented as

$$a = f\left(\sum_{i=0}^N w_i x_i\right)$$

where x_i and w_i are the input data and their weights respectively. It manipulates the presented data through some gradient processing usually gradient descent and afterwards produce an output for the neural network, that contains the parameters in the data [35]. The choice of the activation functions depends on the complexity of the training data and some popular examples are Linear, Sigmoid, Rectified Linear Unit (ReLU), Hyperbolic tangent (Tanh), Leaky ReLU, and Softmax [47]. In modern neural networks, the default recommendation is to use the ReLU defined by the activation function $g(z) = \max[0, z]$ [21]. The core property of most of these activation functions is that they are continuously differentiable.

The goal of machine learning models is to find hyperparameters that minimizes the cost function, and this is done by the optimization process called gradient descent. It is an optimization algorithm that is employed to find the global minimum of a function. It is an iterative process that enables the model to 'learn' by taking the derivative of the loss (cost) function such that it converges toward the minimum value. In a deep neural network, the gradients are typically calculated using the backpropagation process as illustrated in Figure 6. Stochastic gradient descent is an extension of the gradient descent algorithm where a small set (batch) of the training data is used at each step of the 'descent' instead of the entire data [21].

Overfitting is when a model memorizes the noise and fits too closely to the training set. That is, the model/network learns to do well on training set but does not extend this accuracy on validation dataset [60]. Common practices used to prevent overfitting are dropout, early stopping, feature selection, train with more data, and many more.

Overfitting is a serious problem in deep neural networks with large number of parameters. In dealing with overfitting we use dropout as a strategy for preventing neurons from co-adapting by randomly setting a fraction of them to 0 at each training iteration [49].

Momentum is a prominent approach that is used in conjunction with SGD [50]. Instead of depending exclusively on the gradient of the current step to guide the search, momentum considers the gradient of previous steps as well.

2.5.1 Hyperband Algorithm

Hyperband is the state-of-the-art Hyperparameter Optimization (HPO) algorithm that helps to solve the challenges of tuning hyperparameters in machine learning and deep learn-

ing algorithms [4]. Typically, in machine learning algorithms many approaches have been proposed to tackle HPO tuning problems such as Grid Search, Random Search, Bayesian Optimization, Simulated Annealing, Successive Halving, and HyperBand [18]. In recent years, Bayesian Optimization has been employed to improve upon grid and random search approaches but in [28], they found out that Bayesian Optimization operates very well on black box functions but does not work well with parallel resources due because the optimization process is sequential. So they proposed a novel bandit-based approach called Hyperband for HPO. We will briefly explain the concept of successive halving, upon which Hyperband algorithm was developed.

Successive halving randomly sample a set of hyperparameter configurations, then it evaluate the performances of all the currently remaining configurations, after which it throws out the bottom half of the worst scoring configurations, then it evaluate the performances of the configurations again and repeats the same process until one configuration remains [28].

But a huge concern about Successive Halving is finding the right trade-off between total resources and total number of configurations, that is how many configurations is needed at the start and how many cuts should be done. This concern was addressed in the Hyperband Algorithm.

Hyperband is an extension of the successive halving algorithms , that proposed to frequently perform the successive halving method with different budgets to find the best configurations of hyperparameters. It assumes that the best configuration on a large number of iteration should perform in the top half of the configurations after a small number of iterations. This may not always be the case, and situations like this was accounted for by hedging the loop over varying degrees of the aggressiveness , balancing the depth and the

breadth based search. Hyperband requires the ability to sample a hyperparameter configuration, the ability to train them until it reaches a given number of iterations (that is resuming from a previous checkpoint), and get back the loss on a validation set [28]. This is a novel technique that has been adopted for many machine learning algorithms but has also been adopted for deep learning algorithms which often need several configurations of hyperparameters.

3 THE ICE CREAM VENDOR PROBLEM

The ice cream vendor problem is an extension of the Newsvendor problem, whereby a vendor needs to decide how many ordered ice cream is necessary to satisfy his customers (e.g. students on campus) **considering various exogenous factors**, in order to realise an optimum profit (minimum cost) at the end of the period. For the purpose of this thesis, we will limit our scope to a single period.

3.1 Derivation of the Ice Cream Vendor Problem

An ice cream vendor on a college campus buys X units of ice cream at a cost of $\$c$ each. He sells each for $\$p$ and expects a demand D at the end of a selling period. At the end of this period, he incurs an underage cost c_u when the demand D is more than the ordered ice cream X . Otherwise, he incurs an overage cost c_o when the demand D is less than the ordered ice cream X . In a classical newsvendor problem, the demand distribution is known or can be estimated, but in a real-world scenario like that of an ice cream vendor, the **demand doesn't follow a known distribution since it depends on many exogenous features such as day of the week, temperature, wind speed, precipitation, and many more.**

Therefore, our goal is not to predict the demand but to predict the order quantity that minimizes the cost function below, that is, given a set of data points:

$$\begin{aligned} & [(z_1, D_1), (z_2, D_2), \dots, (z_n, D_n)]_{i=1}^n \\ & \underset{w}{\text{minimize}} \quad \frac{1}{n} \sum_{i=1}^n \left(c_u [D_i - f(z_i, w)]^+ + c_o [f(z_i, w) - D_i]^+ \right) \end{aligned} \quad (26)$$

where z are data features (columns), w are weights, D_i are demand, and $f(z, w)$ is a nonlinear relationship between the parameters and their weights. As illustrated in section (2.5), order quantity $X = f(z, w)$ is a composition of functions, which defines relationship between order quantity and features.

3.2 Simulation of Ice Cream Vendor

In order to produce the relevant demand data necessary to solve this problem, in the absence of historical data, we simulated an ice cream vendor using the discrete event simulation technique to produce feature dependent demand data. We set up the simulation to mirror the real life situation of an ice cream vendor that sells ice cream from 10am to 3pm on a college campus with 10,000 students, taking into account how temperature, precipitation, cloud cover, relative humidity could affect the demand for ice cream during the period.

In setting up the simulation in Simpy, we concentrated solely on the consumer (students in this case), and modelled the event base on features that may affect the decision of a consumer to either buy an ice cream or not, such as class schedules, temperature at a particular time, day of the week, and so on. We assumed that

- students take 15 hours of classes each week (i.e 5 courses at 3 hours each)
- each student has a meal plan with the college but often get ice cream or coffee from a vendor
- students have two types of class schedule, one hour class on Monday, Wednesday and Friday , and one and half hours on Tuesday and Thursday.

- a student has a 2 to 1 odds of scheduling classes on Tuesday and Thursday
- classes are from 8am to 4pm from Monday to Friday.

We defined a function that randomly generates a student class schedule and outputs a free time in a day that the student might decide to get ice cream, then we defined another function that outputs ‘1’ if the students gets ice cream and ‘0’ if not. Then we defined functions that outputs multipliers that either increase or decrease the odds of getting an ice cream at that time, these odds are dependent on temperature, cloud cover, relative humidity, precipitation and wind speed, which then randomly generates a meaningful demand.

The model parameters were defined inside a Python class (CollegeStudents) with attributes and multiple instances of schedules were created to generate 25000 rows of feature dependent demand data. All simulations were done on a 12 cores computer with cores of 2.6 GHz computation power and 128 GB of memory. The Simpy code is listed in Appendix A. Table 2 shows a sample of the simulated data.

Table 2: Simulated Demand Data

Day	Tmax	Tmin	FeelsLike	Precip	W.Speed	C.Cover	R.Humidity	Demand
M	67.5	53.4	59.3	0.02	20.4	52.0	63.09	852.0
T	74.1	41.0	42.3	0.00	20.0	1.8	46.99	280.0
T	74.1	41.0	42.3	0.00	20.0	1.8	46.99	280.0
R	75.6	46.8	60.1	0.13	14.2	58.6	66.72	270.0
F	61.0	46.2	42.7	0.02	17.2	68.5	59.96	846.0
M	62.1	31.8	44.9	0.00	9.1	35.2	56.66	916.0
T	59.4	38.9	50.2	0.00	8.6	88.3	62.22	309.0
W	64.0	46.1	46.2	0.00	7.6	87.2	53.36	913.0
R	63.3	46.4	46.1	0.02	10.8	70.6	54.53	310.0
F	72.8	38.5	55.9	0.00	14.8	31.3	56.90	891.

3.3 Linear Programming Approach

The solution of a linear programming problem reduces to finding the optimum value of the linear objective function subject to set of linear constraints. The ice cream vendor problem aims to minimize the sum of the underage and overage cost, in order to predict the optimum order quantity. This can be expressed mathematically as an optimization problem

$$\underset{\beta}{\text{minimize}} \quad \frac{1}{n} \sum_{i=1}^n \left(c_u [D_i - z_i^T \beta]^+ + c_o [z_i^T \beta - D_i]^+ \right) \quad (27)$$

and can be equivalently represented as a linear objective function with linear constraints.

Our goal is to get the β^* for each features that optimizes the order quantity.

$$\begin{aligned} \underset{\beta}{\text{minimize}} \quad & \frac{1}{n} \sum_{i=1}^n (c_u s_i + c_o t_i) \\ & s_i \geq D - z_i^T \beta \\ & t_i \geq z_i^T \beta - D \\ \text{subject to} \quad & s_i \geq 0, t_i \geq 0 \end{aligned} \quad (28)$$

To solve this problem we used the GNU MathProg, the modelling language of the open source GNU Linear Programming Kit (GLPK) project, and calculations are done using the GLP Solve which is embedded in the program. The main components of MathProg are **data, model, and report** [56]. The model is specified in MathProg using model objects such as sets, variables, parameters, constraints, and objectives. Each model object is given a symbolic name that serves as a unique identifier for reference purpose . We used a portion of the simulated data to calculate the parameters β^* , the estimated β^* are used to calculate the order quantities X^* , that is

$$X^* = z_{new}^T \beta^*$$

The MathProg code is listed in Appendix B and Table 3 shows some of the predicted order quantities when the percentage of holding cost to shortage cost was varied.

The columns of Table 3 ‘ P_i Order’ where $[i = 10, 30, 50, 70, 100]$ represent the predicted order quantities when the holding cost is ten percent, thirty percent, fifty percent, seventy percent, and equal to the shortage cost.

Table 3: LP Predicted Order Quantity

Demand	P_{10} Order	P_{30} Order	P_{50} Order	P_{70} Order	P_{100} Order
290	10565	4797	1083	1030	882
922	6667	3655	787	759	676
858	7834	3704	752	715	638
872	8159	4082	958	922	775
996	10383	4769	1057	1006	856
806	9139	4243	1028	985	866
769	6694	3501	874	846	694
334	10297	4952	1012	962	819
280	6255	3384	793	767	624
305	8243	4210	863	826	704
762	8351	4250	1148	1112	952
887	7075	3679	844	814	664
316	11287	5097	1160	1104	944
285	6690	3606	733	704	574
294	9976	4765	1108	1060	935

3.4 Deep Learning Approach

While solution to the classical newsvendor problem requires that we know the demand distribution, we have found that deep learning can be used as an alternative to solve this problem [36], albeit still novel. Referencing [36], they proposed an algorithm based on deep neural network (DNN) that predicts the order quantity that minimizes the vendor’s total

cost.

To solve the ice cream vendor problem, a Pytorch implementation of the deep neural network algorithm in [36] was adapted from Karn Watcharasupat's repository on GitHub (<https://github.com/karnwatcharasupat/DeepNewsvendor>). Referencing [54], their Python implementation which was adapted for our problem could be categorized into six aspects, which are, **data formatting**, **parameter and hyperparameter declaration**, **loss function**, **newsvendor model**, **hyperparameter optimization** (HPO), and **model training**.

3.5 Implementation

For data preparation, they defined a function labelled '*csv_to_npz*' which converts a comma-separated values file (csv) into a numpy file format that compresses array data, then a python class object was created to convert the compressed file into a tensor. A tensor is a multidimensional array of data used in Pytorch . This process aimed to transform our simulated training and testing demand dataset from csv file to tensor format.

They created a python code (params.py) which stores parameter and hyperparameters which would be used to train the newsvendor model, these parameters are minimum and maximum number of hidden layers for our neural network , the number of data features, number of fully connected neural networks, shortage cost and holding cost, number of epochs, batch size, dropout and so on. They created two python classes that defines the newsvendor cost function and its euclidean version as in equation (12) and (14) as explained in section (1.4). These loss functions are used in the backpropagation of the deep learning to update the weights during the SGD and finally outputs the total cost. We compared the two loss values and output the lesser as the best overall cost as seen in Table 4. The ice

cream vendor neural network was adapted from a python class labelled ‘DeepVendorSimple’, which includes a feedforward neural network architecture, with randomized number of hidden layers and nodes; each hidden layer is structured to have function which applies linear transformation (`nn.Linear()`) on the input data, a Leaky ReLU activation function (`nn.LeakyReLU()`) with a default negative slope of 0.01, and dropout (`nn.Dropout()`) with probability 0.2 of an element to be zeroed, which prevents the co-adaptation of neurons.

In order to use SGD with backpropagation of errors to train DNN, a nonlinear activation function that acts like a linear function is needed, allowing complex relationships in the data to be learned. Rectified Linear Unit (ReLU) has been a solution to this problem and has been preferred in modern applications to the sigmoid activation function and hyperbolic tangent activation function [21]. ReLU activation function is a simple calculation that returns the input value, or zero (0) when input is less than or equal to zero (0), that is $g(x) = \max(0, x)$, making it fast and easy to train deep neural networks despite not being differentiable at $x = 0$. In the case where most of the **inputs is in the negative range**, **ReLU tends not to perform because most of its output will be zero** and this affects the gradients flow during backpropagation, which makes large part of the network to become inactive and unable to learn further. This problem is called the ‘dying ReLU’. To avoid such scenario, we used an extension of the ReLU function called Leaky ReLU, which modifies the function to allow a small non-zero gradient when the unit is not active. The leaky ReLU can be defined as

$$LeakyReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{slope} \times x & \text{otherwise} \end{cases} \quad (29)$$

and illustrated in Figure 7.

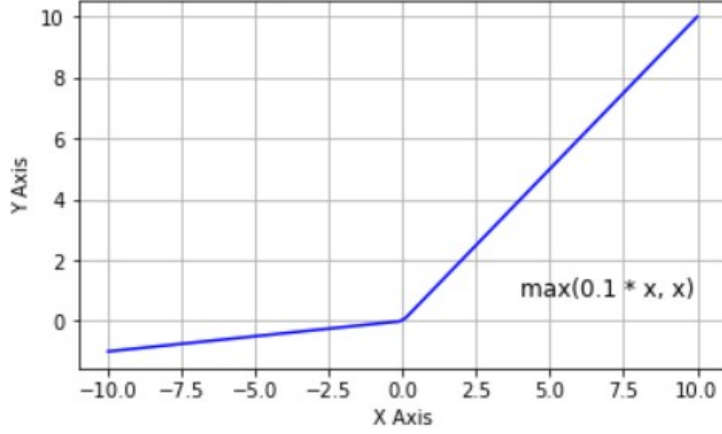


Figure 7: Leaky rectified linear activation function

We adapted the code labelled ‘model.py’ to randomly generate ten (10) neural networks with similar structure as discussed.

For example, Table 5 illustrates the structure of a candidate best network which has five layers, each layers have twelve (12), twenty three (23), forty two (42), twenty two (22) and one (1) nodes respectively.

The hyperparameter optimization phase was implemented with the novel Hyperband algorithm approach as discussed in section (2.5.1), which basically runs multiple iteration of successive halving.

We declared a random number of hidden layers (hidden_layers), number of nodes of ith layer (nn_i), regularization learning rate (lr) and weight decay (lambd).

The number of nodes in the input layer is equal to the number of data features which is twelve (12), but for networks with two hidden layers, we choose $nn_2 \in [0.5nn_1; 3nn_1]$, $nn_3 \in [0.5nn_2; nn_2]$, and $nn_4 = 1$. Similarly, for networks with three hidden layers, we choose $nn_2 \in [0.5nn_1; 3nn_1]$, $nn_3 \in [0.5nn_2; nn_2]$, $nn_4 \in [0.5nn_3; nn_3]$, and $nn_5 = 1$. Then for each network, the number of nodes in each layer are drawn uniformly from the ranges given, the learning rate and weight decay are drawn randomly from $[10^{-5}, 10^{-2}]$. See

column four on Table 4 for the number of layers and nodes for the best network.

Following the defined hyperband, in order to select the best network among the generated ten networks with random structures, we trained each of them for one epoch (25 iterations), obtained the results on the validation set, and then removed the worst 10% of the networks. Subsequently, we run another set of iteration on the remaining networks and remove the worst 10%, and this process was repeated until the best network is obtained. This process is defined in the python code labelled 'train_utils.py' where the the discriminative trainer and discriminative evaluate function was defined, with the EarlyStopping class (MIT licensed open source code) was declared.

Finally, all the defined python codes were imported into one code labelled 'train.py' to be used to train the model and apply all the definitions specified. The result was illustrated in Table 4 with the cost coefficients, the holding cost varied (0.1 to 1.0), and the shortage cost was fixed (1.0), through out several trainings. Note that we assume that the shortage cost is greater than or equal to the holding cost which is nearly almost the case for real world applications.

Table 4: DNN Training Log Table 1

Index	B. Model ID	H. Cost	DNN No. of Layer/Nodes	Min. Cost
0	7	0.1	[12, 29, 50, 32, 1]	335.60
1	1	0.1	[12, 11, 13, 11, 1]	398.78
2	0	0.1	[12, 13, 12, 1]	445.15
3	3	0.2	[12, 18, 16, 1]	648.28
4	8	0.2	[12, 31, 31, 22, 1]	754.18
5	3	0.2	[12, 32, 42, 38, 1]	594.32
6	5	0.3	[12, 31, 50, 27, 1]	841.61
7	6	0.3	[12, 31, 16, 1]	803.81
8	8	0.4	[12, 19, 11, 1]	1280.55
9	5	0.4	[12, 33, 30, 1]	1118.46
10	8	0.4	[12, 15, 19, 18, 1]	942.85
11	1	0.5	[12, 24, 37, 31, 1]	1468.19
12	0	0.5	[12, 21, 31, 16, 1]	1247.30
13	2	0.5	[12, 16, 14, 1]	1693.94

Table 5: DNN Training Log Table 2

Index	B. Model ID	H. Cost	DNN No. of Layer/Nodes	Min. Cost
14	3	0.6	[12, 25, 17, 1]	1338.27
15	1	0.6	[12, 14, 9, 5, 1]	1718.27
16	0	0.6	[12, 33, 25, 1]	1669.87
17	3	0.7	[12, 11, 8, 1]	1382.65
18	2	0.7	[12, 36, 19, 1]	1834.16
19	2	0.7	[12, 28, 21, 1]	1489.31
23	0	0.8	[12, 32, 28, 1]	1768.97
21	1	0.8	[12, 23, 27, 22, 1]	1722.64
22	8	0.8	[12, 28, 27, 25, 1]	1443.54
23	3	0.8	[12, 21, 41, 26, 1]	1776.05
24	8	0.9	[12, 25, 23, 1]	1698.38
25	9	0.9	[12, 28, 15, 1]	1782.36
27	2	1.0	[12, 11, 8, 1]	2165.29
28	3	1.0	[12, 27, 34, 22, 1]	2092.30

3.6 Results and Conclusions

When we compare the predicted order quantities from the deep learning and linear programming approaches, we can infer from Tables 3 **that the linear programming approach is unable to model this type of data and problem.** This is evident from the huge difference between the actual demand and the predicted order quantities. That is, the weights predicted from the linear relationship of the features are not optimum.

However, Table 7 shows that the relationship between the features is better modeled by the functions configured in the DNN, that is, the nonlinearity of the deep neural networks closely modeled the data and was able to closely predict order quantities that are relatively similar to the demand, and could do better with better parameter tuning and network exploration as we can infer from Table 5, that neural networks with higher number of nodes performed better, and have lower cost compared with those with small number of nodes.

Table 6: DNN Test Data

Index	FeelsLike	Precip	W.Speed	C.Cover	R.Humid	Demand
0	87.0	0.0	10.0	52.0	81.0	321.0
1	85.0	0.0	6.0	50.0	82.0	995.0
2	83.0	1.0	13.0	68.0	84.0	916.0
3	87.0	0.0	6.0	37.0	78.0	1056.
4	90.0	0.0	7.0	37.0	80.0	326.0
5	79.0	0.0	7.0	32.0	75.0	347.0
6	68.0	0.0	14.0	34.0	78.0	896.0
7	67.0	0.0	6.0	20.0	74.0	1013.
8	74.0	0.0	11.0	11.0	79.0	335.0

Table 7: DNN Predicted Order Quantity

Demand	P_{10} Order	P_{30} Order	P_{50} Order	P_{70} Order	P_{100} Order
321.0	231.0	216.0	252.0	277.0	302.0
995.0	589.0	687.0	736.0	809.0	824.0
916.0	550.0	655.0	687.0	759.0	781.0
1056.	617.0	697.0	773.0	842.0	875.0
326.0	237.0	221.0	253.0	285.0	305.0
347.0	215.0	219.0	241.0	271.0	299.0
896.0	567.0	638.0	689.0	753.0	808.0
1013.	582.0	667.0	711.0	778.0	831.0
335.0	226.0	666.0	716.0	784.0	821.0

4 FUTURE DIRECTIONS

Ultimately, the purpose of this study is to assist a decision maker in estimating the impact of a supply chain disruption, such as what happens when a student’s schedule changes in the case of an ice cream vendor. **What happens if a pandemic suddenly causes the institution to adopt a hybrid structure?**

The Ice Cream Vendor Problem is perhaps a framework to study **how a sudden shift in demand affects the supply chain** . It could be used in both the corporate and public sectors. For instance, a government may use it as a research tool to determine how a new policy (for example, a change in trade tariffs) would influence the country’s supply chain.

Motivated by the results of the deep learning on the ice cream vendor problem, we suggest that **more exploration can be done to refine the algorithm, such as, increasing the number of generated DNN, increasing the complexity of the simulated dataset by adding some relevant features, and many more.** We suggest that this idea can be extended to other supply chain problems with more complex structure. Finally, other machine learning techniques such as Kernel Density Estimator (KDE), K-Nearest Neighbors (KNN), Random Forest could also be applied on the simulated data and compared to real world datasets.

BIBLIOGRAPHY

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.
- [2] Layth C Alwan, Minghui Xu, Dong-Qing Yao, and Xiaohang Yue. The dynamic newsvendor model with correlated demand. *Decision Sciences*, 47(1):11–30, 2016.
- [3] Taiwo Oladipupo Ayodele. Types of machine learning algorithms. *New advances in machine learning*, 3:19–48, 2010.
- [4] MJ Bahmani. Hyperband and bohb: Understanding state of the art hyperparameter optimization algorithms, May 2021.
- [5] Gah-Yi Ban and Cynthia Rudin. The big data newsvendor: Practical insights from machine learning. *Operations Research*, 67(1):90–108, 2019.
- [6] Dimitris Bertsimas and Nathan Kallus. From predictive to prescriptive analytics. *arXiv preprint arXiv:1402.5481*, 2014.
- [7] Dimitris Bertsimas and Aurélie Thiele. A data-driven approach to newsvendor problems. *Working Papere, Massachusetts Institute of Technology*, 2005.
- [8] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [9] Jennifer Bresnick. What is deep learning and how will it change healthcare?, Dec 2019.

- [10] Real Carbonneau, Kevin Laframboise, and Rustam Vahidov. Application of machine learning techniques for supply chain demand forecasting. *European Journal of Operational Research*, 184(3):1140–1154, 2008.
- [11] CFI. Supply chain - overview, importance, and examples, Jul 2020.
- [12] Boris Defourny. *Machine learning solution methods for multistage stochastic programming*. PhD thesis, PhD thesis, Institut Montefiore, Université de Liège, 2010.
- [13] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.
- [14] Julianna Delua. Supervised vs. unsupervised learning: What’s the difference?, Mar 2021.
- [15] Ayon Dey. Machine learning algorithms: a review. *International Journal of Computer Science and Information Technologies*, 7(3):1174–1179, 2016.
- [16] IBM Cloud Education. What are neural networks?
- [17] Jason Fernando. Supply chain management (scm): What you need to know, Dec 2020.
- [18] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated machine learning*, pages 3–33. Springer, Cham, 2019.
- [19] Robert Fourer, David M Gay, and Brian W Kernighan. *AMPL: A mathematical programming language*. AT & T Bell Laboratories Murray Hill, NJ, 1987.
- [20] Guillermo Gallego and Ilkyeong Moon. The distribution free newsboy problem: review and extensions. *Journal of the Operational Research Society*, 44(8):825–834, 1993.

- [21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [22] Sherry Gray. Inventory optimization for small business.
- [23] Geoffrey Grimmett and Dominic Welsh. *Probability: an introduction*. Oxford University Press, 2014.
- [24] Arthur V Hill. The newsvendor problem. *White Paper*, pages 57–23, 2011.
- [25] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.
- [26] Will Kenton. How supply chains work, Sep 2020.
- [27] Amy Hing-Ling Lau and Hon-Shiang Lau. The newsboy problem with price-dependent demand distribution. *IIE transactions*, 20(2):168–175, 1988.
- [28] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.
- [29] Ben Lorica. Practical applications of reinforcement learning in industry, Dec 2017.
- [30] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR)*.*[Internet]*, 9:381–386, 2020.
- [31] Norm Matloff. Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, 2(2009):1–33, 2008.

- [32] Thomas V Mikosch, Sidney I Resnick, and Stephen M Robinson. Springer series in operations research and financial engineering, 2006.
- [33] Andreas C Müller and Sarah Guido. *Introduction to machine learning with Python: a guide for data scientists.* " O'Reilly Media, Inc.", 2016.
- [34] Steven Nahmias and Ye Cheng. *Production and operations analysis*, volume 6. McGraw-hill New York, 2009.
- [35] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.
- [36] Afshin Oroojlooyjadid, Lawrence V Snyder, and Martin Takáč. Applying deep learning to the newsvendor problem. *IIE Transactions*, 52(4):444–463, 2020.
- [37] Peter O'donovan, Kevin Leahy, Ken Bruton, and Dominic TJ O'Sullivan. Big data in manufacturing: a systematic mapping study. *Journal of Big Data*, 2(1):1–22, 2015.
- [38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [39] Yan Qin, Ruoxuan Wang, Asoo J Vakharia, Yuwen Chen, and Michelle MH Seref. The newsvendor problem: Review and directions for future research. *European Journal of Operational Research*, 213(2):361–374, 2011.

- [40] Cynthia Rudin and Gah-Yi Vahn. The big data newsvendor: Practical insights from machine learning. 2014.
- [41] Tjendera Santoso, Shabbir Ahmed, Marc Goetschalckx, and Alexander Shapiro. A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research*, 167(1):96–115, 2005.
- [42] Herbert E Scarf. A min-max solution of an inventory problem. Technical report, Rand Corp Santa Monica California, 1957.
- [43] Seyda SerdarAsan and Mehmet Tanyas. Dealing with complexity in the supply chain: The effect of supply chain management initiatives. *SSRN Electronic Journal*, 05 2012.
- [44] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczyński. *Lectures on stochastic programming: modeling and theory*. SIAM, 2014.
- [45] Neha Sharma, Reecha Sharma, and Neeru Jindal. Machine learning and deep learning applications-a vision. *Global Transitions Proceedings*, 2(1):24–28, 2021. 1st International Conference on Advances in Information, Computing and Trends in Data Engineering (AICDE - 2020).
- [46] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wangchun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *arXiv preprint arXiv:1506.04214*, 2015.
- [47] P Sibi, S Allwyn Jones, and P Siddarth. Analysis of different activation functions using back propagation neural networks. *Journal of theoretical and applied information technology*, 47(3):1264–1268, 2013.

- [48] Lawrence V Snyder and Zuo-Jun Max Shen. *Fundamentals of supply chain theory*. Wiley Online Library, 2011.
- [49] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [50] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.
- [51] R.S. Sutton and A.G. Barto. *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018.
- [52] Gerhard Tintner. A note on stochastic linear programming. *Econometrica: Journal of the Econometric Society*, pages 490–495, 1960.
- [53] Nazli Turken, Yinliang Tan, Asoo J Vakharia, Lan Wang, Ruoxuan Wang, and Arda Yenipazarli. The multi-product newsvendor problem: Review, extensions, and directions for future research. In *Handbook of newsvendor problems*, pages 3–39. Springer, 2012.
- [54] Karn Watcharasupat. [karnwatcharasupat/deepnewsvendorkarn](https://karnwatcharasupat.github.io/deepnewsvendorkarn/), Feb 2020.
- [55] Martin Wiener, Carol Saunders, and Marco Marabelli. Big-data business models: A critical literature review and multiperspective research framework. *Journal of Information Technology*, 35(1):66–91, 2020.

- [56] Wikibooks. Glpk/gmpl (mathprog) — wikibooks, the free textbook project, 2017. [Online; accessed 25-June-2021].
- [57] Wikipedia contributors. Deep learning — Wikipedia, the free encyclopedia, 2021. [Online; accessed 4-February-2021].
- [58] Wikipedia contributors. Lasso (statistics) — Wikipedia, the free encyclopedia, 2021. [Online; accessed 30-June-2021].
- [59] Wikipedia contributors. Loss function — Wikipedia, the free encyclopedia, 2021. [Online; accessed 11-July-2021].
- [60] Wikipedia contributors. Overfitting — Wikipedia, the free encyclopedia, 2021. [Online; accessed 27-June-2021].
- [61] Wikipedia contributors. Simulation — Wikipedia, the free encyclopedia, 2021. [Online; accessed 5-July-2021].
- [62] Wikipedia contributors. Support-vector machine — Wikipedia, the free encyclopedia, 2021. [Online; accessed 30-June-2021].
- [63] Yanfei Zhang and Junbin Gao. Assessing the performance of deep learning algorithms for newsvendor problem. In *International Conference on Neural Information Processing*, pages 912–921. Springer, 2017.
- [64] Yanju Zhou, Xiaohong Chen, Xuanhua Xu, and Changjun Yu. A multi-product newsvendor problem with budget and loss constraints. *International Journal of Information Technology & Decision Making*, 14(05):1093–1110, 2015.

APPENDICES

Appendix A : Ice Cream Vendor Data Simulation

SimPy code for the discrete event simulation of an Ice cream vendor. The following code

produces daily demand data.

```
#!/usr/bin/env python
# coding: utf-8

# <h1 align='center'> College "Snack" Demand Generation </h1>
#
# Simulated Demands for "daily Ice Cream" snack at a
#   hypothetical "university" with __N_students__ students.

get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt

import numpy as np
import pandas as pd

from numpy.random import choice, rand, uniform

import tqdm.notebook as tqdm

# __Scenario:__ A campus of $N$ students, where each student
# 1. Takes 15 hours of classes ( = 5 courses at 3 hours each)
# 2. Obtains lunch either from the "commons" or from one of a
#   number of local food trucks
# 3. Often gets something "on the fly" from an ice cream or
#   coffee vendor

ScheduleTimes = {('M','W','F'):[8,9,10,11,12,13,14,15,16], #
                  Military time
                  ('T','R'):[8,9.5,11,12.5,14,15.5] } #
                  classes are 1.5 hours on TR

MWF = [8,9,10,11,12,13,14,15,16]
TR = [8,9.5,11,12.5,14,15.5]
_demand = 500 # Assuming No Features
RANDOM_SEED = 42
SIM_TIME = 300 # minutes per day
SIM_START = 10. # hours
SIM_END = 15. # hours
N_students = 10_000
nclasses = 5 # number of courses each student takes
pTR = 2/3 # probability of scheduling a course on a Tuesday or
          Thursday
```

```

class CollegeStudent(object):
    """A template for the features relevant to buying either
        ice cream or
        hot chocolate for a student at a university with $N$
        students"""

    def __init__(self, p_NoFeatures = _demand /N_students ):
        '''Construct the Features that motivate a student to
            buy either icecream or hot chocolate '''
        ## Initialize
        self.p_MWF = p_NoFeatures / (1-pTR)
        self.p_TR = p_NoFeatures / pTR

        ## Build Schedule
        MWF = [8.,9.,10.,11.,12.,13.,14.,15.,16.]
        TR = [8.,9.5,11.,12.5,14.,15.5]
        self.sched = {'MWF':[],'TR':[]}
        for i in range(nclasses):
            if( rand() < pTR and len(self.sched['TR']) < 5 ):
                ## Force at least one MWF course -- if all
                ## courses are TR,
                ## then no times on MWTRF that student goes to
                ## vendor
                crse = choice(TR)
                self.sched['TR'].append(crse)
                TR.remove(crse)
            else:
                crse = choice(MWF)
                self.sched['MWF'].append(crse)
                MWF.remove(crse)
        if( len(self.sched['MWF']) > 0 ): self.sched['MWF'].
            sort()
        if( len(self.sched['TR']) > 0 ): self.sched['TR'].
            sort()

        ## pairs start time and menu choice
        self.FoodOps = dict() # At most 1 per day
        for day in ['M','T','W','R','F']:
            self.FoodOps[day] = []
            daydur = ('MWF',1.0) if day in ['M','W','F'] else
                ('TR',1.5)

            ndurs = len(self.sched[daydur[0]])
            if( ndurs == 0): continue

            ## Food Ops between 10 a.m. and 3 p.m.
            if( self.sched[daydur[0]][0] > SIM_START):
                self.FoodOps[day].append( (SIM_START, self.
                    sched[daydur[0]][0] ) )

            for i in range( ndurs ):
                start_time = max(10,self.sched[daydur[0]][i] +
                    daydur[1])
                if( i+1 < ndurs ):

```



```

        end_time = min(SIM_END, self.sched[daydur
                        [0]][i+1])
    else:
        end_time = SIM_END
    if( start_time < end_time):
        self.FoodOps[day].append( (start_time,
                                    end_time) )

def GetSnackTime(self, day):
    ## Negative if no snack time available that day
    if(len(self.FoodOps[day]) > 0):
        snackperiod = self.FoodOps[day][choice(range(len(
            self.FoodOps[day])))]
        return np.round(uniform(*snackperiod),4)
    else:
        return -1

def BuyIceCream(self, day, FeatureScales = None, ignore =
False ): # does student buy ice cream at time t
    """Student Decides to buy ice cream or not:

    returns Number_purchased """
    if( day in ['T','R'] ):
        p = self.p_TR
    else:
        p = self.p_MWF
    odds = p/(1-p) ## Features increase or decrease the
odds

    if( len(self.FoodOps[day]) == 0):
        return 0 ## Nothing purchased this day
    if( not ignore ): #Ignore all features other than
        what day it is
        ## Features Scale Proportionally
        WinLen = 0
        for per in self.FoodOps[day]:
            WinLen += 60*(per[1] - per[0]) ## in minutes
        odds = odds * WinLen / SIM_TIME ## restricted
        opportunity decreases the odds

        if(np.iterable(FeatureScales) ):
            for scaler in FeatureScales:
                odds *= scaler # scalers are positive (
                    not zero )

    p = odds/(odds + 1) # tranform back to a probability
    if( rand() < p):
        return 1 #buys one ice cream
    else:
        return 0

def __repr__(self):
    return self.sched.__repr__() + '\n' + self.FoodOps.
__repr__()

```

```

# create an instance
#Fred = CollegeStudent()
#Fred.GetSnackTime('M')
#Fred.BuyIceCream('M')
# Weather data for a semester
DailyData = pd.read_csv('ETSU2020-2021AcademicYear.csv',
    index_col=0)
DailyData.head(100)
columns = DailyData.columns

WkDays = ['M','T','W','R','F']*100
DailyData['Day'] = WkDays[:len(DailyData)]
DailyDf = DailyData[['Day','Maximum_Temperature', 'Minimum_Temperature', 'Wind_Chill', 'Heat_Index', 'Precipitation',
    'Wind_Speed', 'Cloud_Cover', 'Relative_Humidity']]
DailyDf.head()

# fill up missing data
DailyDf['Wind_Chill'].fillna(DailyDf['Heat_Index'], inplace =
    True)
DailyDf['Wind_Chill'].fillna(DailyData.Temperature, inplace =
    True)

DailyDf.columns = ['Day','Tmax','Tmin','FeelsLike','HeatIndex',
    'Precipitation','WindSpeed',
    'CloudCover','RelativeHumidity']
DailyDf.drop('HeatIndex',axis=1,inplace=True)
DailyDf
DailyDf.head()

# temperature model
Tmin = 60
Tmax = 80

Am = (Tmax - Tmin)/2
Mn = (Tmax + Tmin)/2
    = np.pi/(60*12) #t in minutes
Temp = lambda t: Mn+Am*np.cos( *(t-17*60))

tran = np.linspace(0,24,1000)
plt.plot(tran,Temp(60*tran))

    = np.pi/(60*12) #time in minutes

# temp at any given time
def TempAtTime(t,tempran):
    Tmin, Tmax = tempran
    Am = (Tmax - Tmin)/2
    Mn = (Tmax + Tmin)/2
    return Mn+Am*np.cos( *(t-17*60)) #max at 5 p.m. (so min
        at 5 a.m.)

IC_ran = (40,110)
#HC_ran = (70,-20)

```

```

# temp to odds
def TempToScaler(temp, ran,      = 0.1 ):
    p = (temp - ran[0])/(ran[1] - ran[0])
    return 1+ *np.tanh(p-0.5)

TempToScaler(100, IC_ran)

# In[ ]:

# features to odds
def PrecipToScaler_IC(precip,      = 0.1):
    return np.exp(- *precip)
def PrecipToScaler_HC(precip,      = 0.1):
    return np.exp( *precip)
def WindSpeedToScaler_IC(windspeed,      = 0.01):
    return np.exp(- *windspeed)
def CloudCoverToScaler_IC(CC,      = 0.1 ):
    p = 1-CC/100
    return 1+ *np.tanh(p-0.5)
def CloudCoverToScaler_HC(CC,      = 0.1):
    p = CC/100
    return 1+ *np.tanh(p-0.5)
def RelativeHumidityToScaler(RH,      = 0.1):
    p = RH/100
    return 1+ *np.tanh(p-0.5)

## Generate Demands -- Single Instance of Schedules
CollegeStudents = [CollegeStudent() for __ in range(N_students
)]
WeekDays = ['M', 'T', 'W', 'R', 'F']

FeaturesAndDemands = DailyDf.copy()
FeaturesAndDemands['Demand'] = np.zeros(len(DailyDf))
pd.DataFrame( columns = [ 'Day', 'Tmax', 'Tmin', 'FeelsLike', '
    Precipitation', 'WindSpeed',
                                'CloudCover', '
                                RelativeHumidity
                                ', 'Demand'
                                ],
                                index = DailyDf.index)

fdmax = np.zeros(6)
fdmin = 1e10*np.ones(6)
for idx, row in tqdm.tqdm(DailyDf.iterrows(), total=DailyDf.
    shape[0]):
    FeatureScalars = np.array(
        [ PrecipToScaler_IC(row.Precipitation)
          ,
          WindSpeedToScaler_IC(row.WindSpeed),
          CloudCoverToScaler_IC(row.CloudCover
          ),
          RelativeHumidityToScaler(row.
          RelativeHumidity),
          TempToScaler(rowFeelsLike, IC_ran),

```

```

1.0 ] ) # for Max, Min temp scaling
Tmax = row.Tmax
Tmin = row.Tmin
n_purchases = 0
for student in CollegeStudents:
    snackTemp = TempAtTime(student.GetSnackTime(row.Day),
        (Tmin,Tmax))
    FeatureScalars[-1] = TempToScaler(snackTemp, IC_ran )
    n_purchases += student.BuyIceCream(row.Day,
        FeatureScales=FeatureScalars)
FeaturesAndDemands.loc[idx,'Demand'] = n_purchases
fdmax = np.maximum(FeatureScalars, fdmax)
fdmin = np.minimum(FeatureScalars, fdmin)
FeaturesAndDemands.head(15)

fdmin, fdmax

FeatureScalars
FeaturesAndDemands.tail(20)

## Multiple Instances of Schedules to create Demands
FeaturesAndDemands = DailyDf.copy()
for i in tqdm.trange(100):
    CollegeStudents = [CollegeStudent() for __ in range(
        N_students)]
    FeaturesAndDemands['Demand%s'%i] = np.zeros(len(DailyDf))
    pd.DataFrame( columns = [ 'Day', 'Tmax', 'Tmin', 'FeelsLike',
        'Precipitation', 'WindSpeed',
        'CloudCover',
        'RelativeHumidity',
        'Demand'
    ],
        index = DailyDf.index)

fdmax = FeatureScalars
fdmin = FeatureScalars
for idx, row in tqdm.tqdm(DailyDf.iterrows(), total=
    DailyDf.shape[0], leave=False):
    FeatureScalars = np.array(
        [ PrecipToScaler_IC(row.
            Precipitation),
            WindSpeedToScaler_IC(row.
                WindSpeed),
            CloudCoverToScaler_IC(row.
                CloudCover),
            RelativeHumidityToScaler(row.
                RelativeHumidity),
            TempToScaler(rowFeelsLike,
                IC_ran),
            1.0 ] ) # for Max, Min temp
                scaling

    Tmax = row.Tmax
    Tmin = row.Tmin
    n_purchases = 0

```

```

for student in CollegeStudents:
    snackTemp = TempAtTime(student.GetSnackTime(row.
        Day), (Tmin,Tmax))
    FeatureScalars[-1] = TempToScaler(snackTemp,
        IC_ran )
    n_purchases += student.BuyIceCream(row.Day,
        FeatureScales=FeatureScalars)
FeaturesAndDemands.loc[idx,'Demand%s'%i] = n_purchases
fdmax = np.maximum(FeatureScalars, fdmax)
fdmin = np.minimum(FeatureScalars, fdmin)

## This Will Take Approximately 3(100) = 300 minutes (5 hours)

FeaturesAndDemands.head(10)

one_hot = pd.get_dummies(FeaturesAndDemands['Day'])

FeaturesAndDemands = FeaturesAndDemands.drop('Day' , axis= 1)

FeaturesAndDemands = FeaturesAndDemands.join(one_hot)
FeaturesAndDemands.head(10)

cols = list(FeaturesAndDemands.columns.values)

```

Appendix B : Solutions to Ice Cream Vendor Problem

Appendix B.1 : Linear Programming Approach

MathProg code for solving the newsvendor problem with implementation on jupyter

notebook.

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np
import pandas as pd
pd.set_option('display.max_rows', None)

# converts dataframe to .dat file
def ToGLPKdat(Df):
    """Df should be a DataFrame
    """
    Lines = 'set ROWS:=\n'
    for row in Df.index:
        Lines += '%s\n' % row
    Lines += ';\nset COLS:=\n'
    for col in Df.columns[0:-1]:
        Lines += '%s\n' % col
    Lines += ';\n\nparam B:\n'
    for col in Df.columns[-1]:
        Lines += '%s\n' % col
    Lines += ':=\n'
    Lines += '\n%s\n' % Df['Demand']
    Lines += ';\n\nparam A:\n'
    for col in Df.columns[0:-1]:
        Lines += '%s\n' % col
    Lines += ':=\n'
    for row in Df.index:
        Lines += '\n%s\n' % row
        for col in Df.columns[0:-1]:
            Lines += '%s\n' % Df.loc[row,col]
    Lines += ';\n\nend;\n'
    with open('Training_Ice.dat','w') as FileObject:
        print('Writing to Training_Ice.dat')
        FileObject.write(Lines)
    return Lines

# model implementation in jupyter notebook
%%script glpsol -m /dev/stdin -d Training_Ice.dat

set ROWS;
set COLS;

# Parameters
```

```

param A{i in ROWS,j in COLS};
param B{i in ROWS};
param c_u :=1.0;
param c_o :=1.0;

# Variables
var s{j in COLS} >=0;
var t{j in COLS} >=0;
var Beta{COLS} >=0;

# Objective function
minimize ExpectedCost:
sum{j in COLS}(c_u*s[j] + c_o*t[j]);

# Constraints
subject to Underage{i in ROWS,j in COLS}:
s[j] >= B[i] - A[i,j]*Beta[j];

subject to Overage{i in ROWS,j in COLS}:
t[j] >= A[i,j]*Beta[j] - B[i] ;

subject to S{j in COLS}:
s[j] >= 0;

subject to T{j in COLS}:
t[j] >= 0;

# solution
solve;

#output
display Beta;
display ExpectedCost;

end;

```

Appendix B.2 : Deep Learning approach

The following code are for implementation of the deep learning solution to the Ice cream vendor problem. The code was adapted from Karn Watcharasupat's repository on GitHub (<https://github.com/karnwatcharasupat/DeepNewsvendor>) and I extend gratitude to the authors for making it available.

```
# parameters declaration
NN_MIN_LAYER = 2
NN_MAX_LAYER = 3
NUM_FEATURES = 12 #changed
DROPOUT = 0.2
NUM_MODELS = 10
SHORTAGE_COST = 1
HOLDING_COST = 1
NUM_EPOCHS = 25
BATCH_SIZE = 32 # 16
ANNEALING_FACTOR = 1e-5
PATIENCE = 10

#data preparation
import numpy as np
import params
import torch
from torch.utils.data import Dataset

class SalesDataset(Dataset):
    # Pytorch dataset for OpenMIC
    def __init__(self, npz_path, randomize = True):
        self.randomize = randomize
        if not self.randomize:
            data = np.load(npz_path)
            self.X = data['X']
            self.Y = data['Y']
            self.length = self.X.shape[0]
        else:
            self.length = 1000

    def __len__(self):
        return self.length

    def __getitem__(self, index):
        if not self.randomize:
            X = self.X[index]
            Y = self.Y[index]
        else:
            X = np.random.rand(100, 24, params.NUM_FEATURES)
            Y = np.random.rand(100, 24)

        X = torch.tensor(X, requires_grad=False, dtype=torch.
```



```

        float32)
    Y = torch.tensor(Y.astype(float), requires_grad=False,
                     dtype=torch.float32)

    return X, Y

# hyperband hyperparameter optimization
import numpy as np
import params

def random_nodes(hidden_layers):

    l_bound = 0.5

    if hidden_layers == 2:
        u_bound = [0.,3.,1.]
    elif hidden_layers == 3:
        u_bound = [0.,3.,2.,1.]
    else:
        raise NotImplementedError

    n_nodes = []
    n_nodes.append(params.NUM_FEATURES)

    for i in range(1, hidden_layers+1):
        n = int(np.round((np.random.rand()*(u_bound[i]-l_bound)
                        ) + l_bound) * n_nodes[i-1]))
        n_nodes.append(n)

    n_nodes.append(1)

    n_nodes = np.array(n_nodes).astype(np.int)

    print(f'Generating network with {hidden_layers+2} layers,
          each with {n_nodes} nodes')

    return n_nodes

def random_nn_params():

    hidden_layers = np.random.randint(params.NN_MIN_LAYER,
                                       params.NN_MAX_LAYER+1)
    n_nodes = random_nodes(hidden_layers)
    lr = np.power(10, np.random.rand()*3. - 5.)
    lambd = np.power(10, np.random.rand()*3. - 5.)

    return hidden_layers, n_nodes, lr, lambd

def __sanity_check():
    print(random_nn_params())

__sanity_check()

```

```

# define feedforward dnn
import torch
import torch.nn as nn
import numpy as np
import params

def generate_fc(n_hidden, n_nodes):
    layers = []

    in_chan = n_nodes[0]
    for i in range(n_hidden + 2):
        out_chan = n_nodes[i]
        layers += [
            nn.Linear(in_chan, out_chan),
            nn.LeakyReLU(),
            nn.Dropout(params.DROPOUT)
        ]
        in_chan = out_chan

    return nn.Sequential(*layers)

class DeepVendorSimple(nn.Module):

    def __init__(self, model_type = 'simple_fc', n_hidden = 2,
                 n_nodes = [4,3,2,1]):
        super().__init__()

        self.n_features = n_nodes[0]

        if model_type == 'simple_fc':
            self.net = generate_fc(n_hidden, n_nodes)
        else:
            raise NotImplementedError

    def forward(self, x):
        '''
        input    x:      size n_product by n_obs by n_features
        output   y:      size n_product
        '''
        y = self.net(x)
        y = y.squeeze()

        return y

# euclidean loss function
import torch
import torch.nn as nn
import numpy as np

class EuclideanLoss(nn.Module):

    def __init__(self, c_p, c_h):
        super().__init__()

```

```

        self.c_p = c_p
        self.c_h = c_h

    def forward(self, y, d):
        """
        y: prediction, size = (n_product, n_obs)
        d: actual sales, size = (n_product, n_obs)
        """

        diff = torch.add(y, -d)
        diff = torch.add(torch.mul(torch.max(diff, torch.zeros(
            1)), self.c_p), torch.mul(torch.max(-diff, torch.
            zeros(1)), self.c_h))
        diff = torch.norm(diff)
        diff = torch.sum(diff)
        return diff

# linear loss function
class CostFunction(nn.Module):

    def __init__(self, c_p, c_h):
        super().__init__()
        self.c_p = c_p
        self.c_h = c_h

    def forward(self, y, d):
        """
        y: prediction, size = (n_product, n_obs)
        d: actual sales, size = (n_product, n_obs)
        """

        cost = torch.add(y, -d)
        cost = torch.add(torch.mul(torch.max(cost, torch.zeros(
            1)), self.c_p), torch.mul(torch.max(-cost, torch.
            zeros(1)), self.c_h))
        cost = torch.sum(cost)

        return cost

# model evaluation
import os
import errno
import shutil
from tqdm import tqdm
import numpy as np
import torch

class AverageMeter(object):
    """Computes and stores the average and current value"""

    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0

```

```

        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count

def discriminative_trainer(model, data_loader, optimizer,
                           criterion):
    torch.cuda.synchronize()
    # print(torch.cuda.memory_allocated())
    # model.eval()
    model.train()
    loss_tracker = AverageMeter()

    for (X, Y) in tqdm(data_loader):
        torch.cuda.empty_cache()
        # device = torch.device('cuda:0' if torch.cuda.
        # is_available() else 'cpu')
        device = torch.device('cpu')
        X = X.to(device)
        Y = Y.to(device)
        outputs = model(X)
        loss = criterion(outputs, Y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        loss_tracker.update(loss.item())
    return loss_tracker.avg

def discriminative_evaluate(model, data_loader, criterion):
    torch.cuda.empty_cache()
    # device = torch.device('cuda:0' if torch.cuda.
    # is_available() else 'cpu')
    device = torch.device('cpu')
    model.eval()
    loss_tracker = AverageMeter()
    for (X, Y) in tqdm(data_loader):
        X = X.to(device)
        Y = Y.to(device)
        outputs = model(X)
        result = outputs.detach().numpy()
        loss = criterion(outputs, Y)
        loss_tracker.update(loss.item())
    return loss_tracker.avg, result

# earling stopping for hyperband
class EarlyStopping:

```

```

# MIT License

# Copyright (c) 2018 Bjarte Mehus Sunde

# Permission is hereby granted, free of charge, to any
# person obtaining a copy
# of this software and associated documentation files (the
# "Software"), to deal
# in the Software without restriction, including without
# limitation the rights
# to use, copy, modify, merge, publish, distribute,
# sublicense, and/or sell
# copies of the Software, and to permit persons to whom
# the Software is
# furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice
# shall be included in all
# copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF
# ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
# MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN
# NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
# DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
# OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
# OTHER DEALINGS IN THE
# SOFTWARE.
"""Early stops the training if validation loss doesn't
improve after a given patience."""

def __init__(self, patience=7, verbose=False):
    """
    Args:
        patience (int): How long to wait after last time
            validation loss improved.
            Default: 7
        verbose (bool): If True, prints a message for each
            validation loss improvement.
            Default: False
    """
    self.patience = patience
    self.verbose = verbose
    self.counter = 0
    self.best_score = None
    self.early_stop = False
    self.val_loss_min = np.Inf

def __call__(self, val_loss, model):

```

```

score = -val_loss

if self.best_score is None:
    self.best_score = score
    # self.save_checkpoint(val_loss, model)
elif score < self.best_score or np.abs(score - self.
best_score) < 1e-4:
    self.counter += 1
    print(
        f'EarlyStopping counter: {self.counter} out of
        {self.patience}')
    if self.counter >= self.patience:
        self.early_stop = True
else:
    self.best_score = score
    # self.save_checkpoint(val_loss, model)
    self.counter = 0

def save_checkpoint(self, val_loss, model):
    '''Saves model when validation loss decrease.'''
    if self.verbose:
        print(
            f'Validation loss decreased ({self.
            val_loss_min:.6f}-->{val_loss:.6f}).
            Saving model...')
    torch.save(model.state_dict(), 'checkpoint.pt')
    self.val_loss_min = val_loss

## model training and best model selection
import random
import torch
import numpy as np
import os
from copy import deepcopy
import datetime
from tqdm.notebook import tqdm, trange, trange
from torch.utils.tensorboard import SummaryWriter
import params
from train_utils import *

model_type = 'simple_fc' # 'att_context'
id = 'simple_fc'

# Set hyperparams:
missing = False
num_epochs = params.NUM_EPOCHS
batch_size = params.BATCH_SIZE
anneal_factor = params.ANNEALING_FACTOR
patience = params.PATIENCE

seed = np.random.randint(0,100000) ## change seed

torch.cuda.synchronize() # comment
torch.cuda.empty_cache()

```

```

# set random seeds
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
np.random.seed(seed)

# Other imports now
from data_utils import SalesDataset
from torch.utils.data import DataLoader

TRAIN = 'train.npz'
TEST = 'test.npz'

train_data = SalesDataset(TRAIN, randomize = False)
test_data = SalesDataset(TEST, randomize = False)

train_loader = DataLoader(train_data, batch_size, shuffle =
    True)
test_loader = DataLoader(test_data, batch_size, shuffle = True
    )
val_loader = test_loader

#-----#
# Model Definition #
#-----#

from hyperband import random_nn_params
from model import DeepVendorSimple
from loss_function import EuclideanLoss, CostFunction

best_overall_model = None
best_overall_cost = 100000.
best_model_idx = 0

for model_idx in tnrange(params.NUM_MODELS):
    hidden_layers, n_nodes, lr, weight_decay =
        random_nn_params()

    model = DeepVendorSimple(n_hidden = hidden_layers, n_nodes
        = n_nodes)

    torch.cuda.empty_cache()
    #torch.cuda.memory_summary()
    #device = torch.device('cuda:0' if torch.cuda.is_available
        () else 'cpu')
    device = torch.device('cpu')
    print(device)
    model = model.to(device)

    optimizer = torch.optim.Adam(
        model.parameters(),
        lr=lr,
        weight_decay=weight_decay)

```

```

criterion = EuclideanLoss(params.SHORTAGE_COST, params.
    HOLDING_COST)
test_criterion = CostFunction(params.SHORTAGE_COST, params
    .HOLDING_COST)

writer_path = os.path.join(id+'_idx_'+str(model_idx))
writer = SummaryWriter(writer_path)

best_model = None
best_val_loss = 100000.0

try:
    early_stopping = EarlyStopping(patience=patience,
        verbose=False)

    for epoch in tnrange(num_epochs, total = None): ### (
        range):
        torch.cuda.empty_cache()

        # Train model
        loss = discriminative_trainer(
            model=model,
            data_loader=train_loader,
            optimizer=optimizer,
            criterion=criterion)
        #print(f'epoch: {epoch}, loss: {loss}')

        # log in tensorboard
        writer.add_scalar('Training/Prediction_Loss', loss
            , epoch)

        # Eval model
        loss, result = discriminative_evaluate(model,
            val_loader, test_criterion)
        writer.add_scalar('Validation/Prediction_Cost',
            loss, epoch)

        if loss < best_val_loss:
            best_val_loss = loss
            best_model = deepcopy(model)

        # Anneal LR
        early_stopping(loss, model)
        if early_stopping.early_stop: # and epoch >
            params.MIN_EPOCH:
                #print("Early stopping")
                break

except KeyboardInterrupt:
    print('Stopping_training.Now_testing')

# Test the model
model = best_model
torch.cuda.empty_cache()
loss,result= discriminative_evaluate(model, test_loader,

```



```
        test_criterion)
    print('Test_Prediction_Cost:', loss)
    if loss < best_overall_cost:
        best_overall_model = deepcopy(model)
        best_overall_cost = loss
        best_model_idx = model_idx
        best_model_nodes = n_nodes
    torch.save(model.state_dict(), os.path.join('best_model.pth'))
```

VITA

GAFFAR OLAMIDE SOLIHU

Education: B.S. Mathematics, University of Ilorin,
Ilorin, Nigeria 2014
M.S. Mathematical Sciences, East Tennessee State University,
Johnson City, Tennessee 2021

Professional Experience: High School Teacher , Salaudeen Community School,
Lagos State, Nigeria, 2015–2016
College PT Instructor, Lagos State Polytechnic,
Lagos State, Nigeria, 2017–2018
Mathematic Tutor, East Tennessee State University,
Johnson City, Tennessee, 2019–2021
Graduate Assistant, East Tennessee State University,
Johnson City, Tennessee, 2019–2021