MACHINE LEARNING BASED TRUNCATION POINT ESTIMATION IN STEADY-STATE SIMULATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BURAK GIRIT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

SEPTEMBER 2023

Approval of the thesis:

MACHINE LEARNING BASED TRUNCATION POINT ESTIMATION IN STEADY-STATE SIMULATION

submitted by **BURAK GIRIT** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Pelin Bayındır
Head of Department, **Industrial Engineering** _____

Prof. Dr. Nur Evin Özdemirel
Supervisor, **Industrial Engineering, METU** _____

Prof. Dr. Pınar Karagöz
Co-supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Mustafa Kemal Tural (Chair)
Industrial Engineering, METU _____

Prof. Dr. Nur Evin Özdemirel
Industrial Engineering, METU _____

Prof. Dr. Pınar Karagöz
Computer Engineering, METU _____

Prof. Dr. Hakkı Toroslu
Computer Engineering, METU _____

Assist. Prof. Dr. Derya Dinler
Industrial Engineering, Hacettepe University _____

Date:08.09.2023

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Burak Girit

Signature :

ABSTRACT

MACHINE LEARNING BASED TRUNCATION POINT ESTIMATION IN STEADY-STATE SIMULATION

Girit, Burak
M.S., Department of Industrial Engineering
Supervisor: Prof. Dr. Nur Evin Özdemirel
Co-Supervisor: Prof. Dr. Pınar Karagöz

September 2023, 128 pages

In this study, we focus on estimating the truncation point for solving the initialization bias problem encountered in output analysis for steady-state simulations by using machine learning methods.

Since the initial conditions of simulation do not represent the steady-state, biased data originating from the initial state must be eliminated in order to analyze the simulation output properly. A truncation point in simulation output data has to be determined for eliminating this initialization bias. In this study, the truncation point estimation capabilities of multilayer perceptron regressor, long short-term memory, and conditional recurrent neural networks are investigated.

In order to train these three neural networks and test their performances, the second order autoregressive model and M/M/1 queueing system model are used to generate data representative of the simulation output. Moreover, these three machine learning methods are compared with the conventional truncation estimation methods MSER and MSER-5.

Experimental results show that the multilayer perceptron regressor network has superior performance compared to other methods, in terms of truncation point estimation error and coverage of the confidence intervals for steady-state expected values. However, the long short-term memory and conditional recurrent neural networks cannot learn effective truncation point estimation with the network configurations used.

Keywords: initialization bias in steady-state simulation, truncation point estimation, multilayer perceptron regressor (MLPR), long short-term memory (LSTM), conditional recurrent neural network (CRNN), machine learning

ÖZ

KARARLI DURUM SİMÜLASYONUNDA MAKİNE ÖĞRENMESİ TABANLI KESME NOKTASI TAHMİNLENMESİ

Girit, Burak
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Nur Evin Özdemirel
Ortak Tez Yöneticisi: Prof. Dr. Pınar Karagöz

Eylül 2023 , 128 sayfa

Bu çalışmada, kararlı durum simülasulasyonunda karşılaşılan başlangıç sapması problemini çözebilmek amacıyla, makine öğrenmesi teknikleri kullanılarak kesme noktası tahminlemeye odaklanılmıştır.

Simülasyonda başlangıç koşulları kararlı durumu temsil etmediğinden, simülasyon çıktılarının sağlıklı bir şekilde analiz edilebilmesi için, başlangıç durumundan kaynaklanan sapmalı veriler elenmelidir. Bu başlangıç sapmasını elemek için, simülasyon çıktısında bir kesme noktası belirlenmesi gerekir. Bu çalışmada, çok katmanlı algılayıcı regresör, uzun kısa dönemli hafıza ve koşullu yinelenen sinir ağlarının kesme noktası tahminleme yetenekleri araştırılmıştır.

Bu üç tip sinir ağının eğitilmesi ve performanslarının test edilmesi amacıyla kullanılmak üzere, simülasyon çıktılarını temsil eden verileri üretmek için, ikinci derece özbağlanımlı model ve M/M/1 kuyruk sistemi modeli kullanılmıştır. Ek olarak, bu üç makina öğrenmesi temelli yöntem, bilinen kesme noktası tahminleme yöntemleri

olan MSER ve MSER-5 ile karşılaştırılmıştır.

Deneysel sonuçlara göre, kesme noktası tahminleme hatası ve kararlı durum beklendik değerleri için kurulan güven aralıklarının bu beklendik değerleri kapsaması dikkate alındığında, çok katmanlı algılayıcı regresör ağı diğer yöntemlerden kayda değer daha iyi performans göstermiştir. Buna karşılık, uzun kısa dönemli hafıza ve koşullu yinelenen sinir ağları, belirlenen ağ konfigürasyonları ile, etkin kesme noktası tahminlemesi yapmayı öğrenememiştir.

Anahtar Kelimeler: kararlı durum simülasyonunda başlangıç sapması, kesme noktası tahmini, çok katmanlı algılayıcı regresör, uzun kısa dönemli hafıza, koşullu yinelenen sinir ağı, makina öğrenmesi

To my beloved family

ACKNOWLEDGMENTS

First of all, I would like to thank Prof. Dr. Nur Evin Özdemirel and Prof. Dr. Pınar Karagöz for the time and guidance they devoted to this study for almost a year and a half. This study could not have been completed without their contributions.

Additionally, I would like to express my gratitude to the members of the jury committee, Assoc Prof. Dr. Mustafa Kemal Tural, Prof. Dr. Hakkı Toroslu and Assist. Prof. Dr. Derya Dinler, for reviewing this study and contributing with their comments.

Finally, I am grateful to my family and all my loved ones for their support and motivation. Feeling their positive attitude and trust in me gave me strength while carrying out this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

MLP	Multilayer Perceptron
MLPR	Multilayer Perceptron Regressor
LSTM	Long Short-Term Memory
CRNN	Conditional Recurrent Neural Network
MSER	Marginal Standard Error Rules
AP	Autoregressive Model Parameters
BT	Bias Type
BC	Bias Coefficient
TP	Truncation Point
DS	Data Smoothing

CHAPTER 1

INTRODUCTION

Steady-state simulation is a fundamental modeling tool used to analyze the behavior of a simulated system when it reaches equilibrium. As a result of using simulation to study the steady-state behavior of systems, more realistic and effective decisions can be made since simulation facilitates prediction of the future as well as evaluation of alternative strategies. Therefore, steady-state simulations are utilized for various purposes, such as process optimization (Kim et al., 2009), queueing system improvement (Glynn and Iglehart, 1988), and traffic management (Chinyere et al., 2011).

Steady-state simulations do not have known starting conditions and do not need a natural event to end the simulation. However, some time is required for these simulations to reach the steady-state. The states in which simulations are started generally do not represent the steady-state of the system and therefore behave differently from normal until the steady-state of the simulation model is reached (Banks et al., 2005). The part of a simulation run that has not yet reached the steady-state because of the arbitrary initial conditions is called the transient state (Law, 2015). In order to analyze the system behavior in steady-state, the simulation outputs obtained during the transient state must be eliminated. Otherwise, the transient state creates a bias in the final simulation outputs. This problem is called the initialization bias problem in the literature.

In order to eliminate the initial bias for the steady-state analysis to be more accurate, the output values that are obtained during the transient state and cause the initial bias should be truncated from the output data to be used in the analysis. When the simulation output is considered as a time series, the point in this time series after which the initial bias becomes insignificant is called the truncation point. The rest of the time

series that come after the truncation point can be assumed to represent the steady-state. Although there is no exact method of detecting the truncation point in every simulation, there are many different studies in literature that propose approximate truncation point detection methods and compare these methods. Hoad et al. (2008) provide a review of 42 different truncation point estimation methods. Welch's graphical method (Welch, 1983), MSER and MSER-5 (White et al., 2000), batch means test (Cash et al., 1992), and Schruben's test for initial bias (Schruben, 1982) are some of the well known conventional methods that are used for eliminating the initial bias.

Machine learning techniques, which have become more popular as the technology developed, are often used in many different areas, such as pattern recognition (Orriols-Puig and Bernadó-Mansilla, 2008), product recommendation (Chen et al., 2017), disease diagnosis (Kourou et al., 2015), and stock market price prediction (Shen et al., 2012). Moreover, Giabbanelli (2019) discusses several examples of how simulation modeling and machine learning techniques can be used together to solve each other's known problems. Even though machine learning methods and simulation models have recently been utilized together more frequently, estimating the truncation point in steady-state simulations is an understudied topic in this area. To the best of our knowledge, there exists only one study in the literature utilizing an artificial neural network to estimate the truncation point (Lee and Kyung, 1997).

Inspired by these studies, this thesis focuses on estimating the truncation point to overcome the initialization bias problem by means of machine learning methods. For this purpose, studies were carried out with three types of neural networks, namely Multilayer Perceptron Regressor (MLPR), Long Short-Term Memory network (LSTM), and Conditional Recurrent Neural Network (CRNN). In these studies, the second order autoregressive (AR) model and the M/M/1 queueing system model are used as representative simulation models. Various time series data generated from these models are given as input to the neural networks for training and testing purposes through cross validation. In addition, separately generated test data are used to construct confidence intervals for the known steady-state expected values of the AR and M/M/1 time series, by taking the portion of these time series that comes after the estimated truncation point. The motivation behind this test is that, if the truncation point is estimated accurately, the confidence interval constructed using the data that come after

this truncation point should cover the steady-state expected value.

The results of the machine learning methods are compared with the conventional MSER and MSER-5 methods of truncation point estimation, which are widely used and known to be relatively more successful. We have shown that the MLPR network can make more successful truncation point estimations than MSER and MSER-5 methods, with the network configuration we have determined. In some of the experiments (in general with M/M/1 data), MSER and MSER-5 have difficulty in estimating the truncation point accurately. The MLPR network, on the other hand, only underperformed in high traffic intensity cases of the M/M/1 model. However, this problem is eliminated by using a longer series of M/M/1 output and applying the batch means method.

Although the MLPR network was found to be successful in estimating the truncation point, successful results were not obtained in the experiments with the LSTM and CRNN networks.

We aim to contribute to the literature as the subject of this thesis is an understudied topic. Moreover, thanks to the machine learning methods used for truncation point estimation, it is possible to get encouraging results from the proposed solution approaches, especially the MLPR network. So far in literature, simulation modeling and machine learning techniques are used together mostly for the purposes of statistical input or output analysis. This study demonstrates that using machine learning techniques with simulation models can be beneficial for solving the initial bias problem as well.

The remainder of the thesis is organized as follows.

In the second chapter, studies in the literature about the four topics related to this thesis are reviewed. First of all, the studies on the initialization bias problem in steady-state simulation are discussed. Then, conventional truncation point estimation methods to eliminate the initial bias problem are reviewed. Next, the cooperation between machine learning and simulation, of which this study is a part, and studies on how they are utilized together are described. Finally, prediction or estimation applications of machine learning methods used in solution approaches of this thesis are

overviewed.

In the third chapter, firstly, the initial bias problem, which we are trying to solve, is defined. Then, the AR and M/M/1 queueing system models we use as representative simulation models to generate our datasets, which are also used in other studies on this problem, are described.

In Chapter 4, the machine learning methods we use in our solution approach are briefly explained. Then, the network architectures and parameter configurations we work with in adapting these methods for our problem are given. We first focus on the MLPR network, then the LSTM network, and finally the CRNN.

In Chapter 5, the experiments and the results of these experiments will be discussed. Firstly, generation of datasets is explained, followed by the experimental settings. Afterwards, the performance measures used in evaluating results of the experiments are defined. After explaining how the architecture and hyperparameters of the MLPR, LSTM, and CRNN networks are are determined, the results of the experiments with these networks are presented in terms of the performance measures. Finally, the test results of the MLPR network experiments are compared with results of the MSER and MSER-5 methods, which is often used in literature to overcome the initial bias problem.

In the last chapter, a brief summary of the results and findings of this study are given.

CHAPTER 2

LITERATURE REVIEW

In this chapter, the literature on topics related with this thesis study are reviewed. The initialization bias problem in steady-state simulation is discussed, and some studies dealing with this problem are reviewed in Section 2.1. Conventional truncation point estimation heuristics available in literature are described in Section 2.2. Section 2.3 is allocated to studies on integration of machine learning methods with simulation analysis. Finally, some prediction or estimation applications of machine learning methods used in this study are overviewed in Section 2.4.

2.1 Initialization Bias Problem in Steady-State Simulation

From the perspective of output analysis, Law (2015) classified simulation models as terminating and non-terminating simulations. In his book, terminating simulations are defined as: "A terminating simulation is one for which there is a "natural" event E that specifies the length of each run (replication)." For example, in combat simulations, the end condition of the simulation can be set to stop the simulation if one of the two opposing units becomes inoperative. As soon as this condition is met in the simulation model, the simulation stops and the results can be examined. While the outputs of terminating simulations are analyzed, the initial state of the simulation has a critical role in the analysis as it directly affects the results, and necessary studies are carried out according to the different scenarios of the initial state.

In the same book, Law (2015) described non-terminating simulations as: "A non-terminating simulation is one for which there is no natural event E to specify the length of a run. This often occurs when we are designing a new system or changing

an existing system, and we are interested in the behavior of the system in the long run when it is operating normally." Queueing model simulations are one of the most well-known examples of these non-terminating simulation models. In non-terminating simulation models, the simulation has to progress for a certain amount of time in order for the simulation to reach the steady-state because, usually, the state from which the simulation is started does not represent the characteristics of the steady-state.

Concerning this issue, Currie and Cheng (2016) state: "It is possible to observe an initial period where the output is highly variable before the data series settles down to what can be regarded as its steady-state behavior. This behavior at the start of the simulation run is often termed the initial transient." The authors continue as: "The most common way of dealing with the initial transient is to delete the output from this period, which we define to be the warm up period." In order to analyze the steady-state behavior, the effects of the initial state must be eliminated. This is called the initialization bias problem in the simulation literature.

There are many studies in the literature about the effects of the initialization bias problem, which is the subject of this thesis.

Kim et al. (2018) worked on hydrologic simulations with different initial values of soil moisture conditions and rainfall amounts to analyze the duration of the initial bias. Their research concluded that the time required for the model to reach its steady-state depends on the initial state for the hydrologic simulation models.

Sandıkcı and Sabuncuoglu (2006) presented a similar perspective by analyzing the behavior of the simulation model before reaching the steady-state. In their study, they examined the factors affecting the duration of reaching steady-state in a simulation model developed for a production system.

Grassman (2008) approached this problem from a different perspective than most studies in the literature. In his work, he discusses how the initial state should be set to avoid this problem rather than focusing on a method to eliminate the initiation bias: "If one starts in a state with a high equilibrium probability, one should not use any warm-up period." However, it is not easy to calculate these probabilities for complex systems. For these cases, he proposes to run a pilot simulation and produce a solution

based on the results.

Kolahi (2011) constructed a queueing system simulation model to represent the cellular CDMA system. While explaining the details of the simulation model in his work, he mentions that the initialization bias significantly affects the results. Therefore, this subject must be studied in order to analyze the results of the simulation accurately.

Kelton and Law (1983) explain that the slope values of simulation outputs approaches zero when the simulation model reaches steady-state. They propose that, by considering the slopes of simulation outputs, it is possible to find the point where a simulation model reaches steady-state. After the determination of this point, previous observations can be deleted, and analyses can be made excluding the bias resulting from the initial state.

Law (2015, Chapter 9) extensively discusses results of an experiment where initial bias is not truncated in an attempt for estimating steady-state mean of an output performance measure. According to results of the experiment, when a confidence interval is constructed by ignoring the initial bias, the actual coverage of the confidence interval for the steady-state mean performance can be much lower than the aimed confidence level. This carries a much higher confidence perception in the estimation than there actually is. For example, the analyst may think that the confidence interval covers the true mean with a probability of 0.90 whereas this probability can actually be as low as 0.50 or even lower. Hence, one may end up with extremely unreliable estimations when the initial bias is not properly truncated.

2.2 Conventional Truncation Point Estimation Methods

The truncation point, which is an essential concept in the initiation bias problem, can be defined as the observation value at which the transient phase of the simulation model ends, and the model now reaches the steady-state. Swamidass (2000) gave the definition of truncation point as the following: "The observation beyond which data collection is started is called the truncation point. The objective is to allow the system to 'warm up' and to initiate data collection at the end of the warm-up period." In order to make an accurate and healthy analysis of the simulation output, the truncation point

is determined, and the observation values before this point are not used in the analysis. However, there exists a tradeoff when determining the truncation point. While one should choose a later point in the observations so that there is almost no bias left in the remaining data, it is also necessary to have sufficient data left for the analysis. Because otherwise, the statistical estimates found using limited data would have a large variance. Therefore, it is desirable to find the minimum truncation point that will eliminate significant portion of the initial bias.

There exist many truncation point estimation methods available in literature to overcome the initialization bias problem. Hoad et al. (2008) presented a comprehensive review of the truncation point methods and provided their references in their study. In this work, 42 different methods found in literature are classified into five different groups, as Robinson (2002) also suggested. These groups are:

- Graphical methods
- Heuristic approaches
- Statistical methods
- Initialization bias tests
- Hybrid methods

Another representation of this classification together with specific methods available in each group can be found in Figure 2.1.

In order to overcome the initialization bias problem with relative ease of implementation, Hoad et al. (2008) also studied an automated procedure to determine the truncation point in the simulation output data. The graphical methods, and most of the heuristic and statistical methods were not eligible for automation. Therefore, those methods were not included in the preliminary testing process. Moreover, neither initialization bias tests nor hybrid methods were used. From the heuristic approaches, Mean Squared Error Reduction with a batch size of five (MSER-5), Kimbler's Double Exponential Smoothing method, and Euclidean Distance Method (ED) were taken into account for automation. From the statistical methods, the goodness of fit test, the

Figure 2.1: Truncation Point Estimation Methods (Hoad et al. 2008)

algorithm for a static data set (ASD), and the algorithm for a dynamic data set (ADD) were also included. However, after the preliminary testing, only the MSER-5 method was selected to continue with further testing.

Welch (1983) proposes a graphical method with multiple simulation model replications to determine the truncation point. To briefly summarize this method, Welch's procedure includes taking the averages of individual observation values over replications and then applying moving averages centered at the observation numbers. After plotting this averaged-process data, the truncation point can be determined as the point at which initialization bias is no longer observed in the data.

In the study by White (1997), it is proposed to select a truncation point that minimizes the width of the confidence interval about the truncated sample mean, as the earlier observation values result in increasing the half-width value of the confidence interval since their values are not sufficiently close to the steady-state mean. This method is defined as the Marginal Confidence Rule (MCR) and constitutes the basis of MSER rule.

Robinson (2002) proposed a new approach based on the statistical process control method. The batch means method is employed in creating a control chart since outputs of a simulation model are usually highly autocorrelated and potentially non-normal. The proposed method is tested on seven datasets generated by two different data models (the first-order autoregressive model and the M/M/1 model) with different parameter values.

Oh and Park (2015) suggested a new heuristic method (Exponential Variation Rate, EVR) in order to overcome the initialization bias problem. They compared their method with the MSER-5 method to evaluate their effectiveness, consistency, and confidence. In this comparison, they used the M/M/1 model with four different traffic intensities. Both methods showed successful performances in mean values estimations after truncation point predictions. However, the sample variance values for EVR were higher than the MSER-5 method because the truncation point estimation values of the EVR method were lower than the MSER-5 method.

Most of the studies on the initial bias problem review and compare the methods pro-

posed to eliminate the initial bias and report their performances. For example, Linton and Catherine (2002) compared Welch's method (Welch 1983), the Relaxation Time Heuristic (Roth 1994), Kelton and Law's method (Kelton and Law 1983), and the Marginal Confidence Rule (White 1997) by using two different 2-machine flow shop models. They concluded that among the examined methods, Welch's Method and the Relaxation Time Heuristic appear to offer practical advantages. Welch's Method, in particular, is notable for its ability to address initialization bias without relying on assumptions about the modeled system. This suggests that it could be a valuable approach in elimination of initialization bias.

Mahajan and Ingalls (2004) compared the performances of six different methods that can be used to determine the length of the warmup period on a simple job shop model. Welch's Method, Conway Rule, Statistical Process Control Method, Crossing the Means Rule, MSER-5 Rule, Randomization Test are compared with three model types considering sample mean and variance estimations. Based on the outcomes, it is concluded that no single method demonstrates consistent effectiveness across all model types. Certain methods prove effective for systems with low utilization rates, while others exhibit performance gains for longer run lengths as opposed to shorter ones.

Moreover, Robinson and Ioannou (2007) compared 24 truncation point estimation methods and evaluated them by considering simplicity, ease of implementation, accuracy, required assumptions, and the number of parameters need to be set by scoring their performances from 1 to 5. The strengths and weaknesses of each method have been examined. As a result, no method was identified as the optimal choice.

White et al. (2000), which is the most inspired study for this thesis work, compared five different truncation point methods based on the estimated sample mean and estimated standard deviation, the difference between estimated mean and true mean, the two-sample t-test for equality of means, computational time, and truncation point estimates. MSER, MSER-5, and three different bias detection tests were compared, and the MSER-5 method was the favorable method for discarding the effects of the startup problem.

In another study using the MSER-5 method, Mokashi et al. (2010) compared N-skart

and MSER-5 methods. They concluded that although the N-skart method performs better, it is a more complicated method in terms of implementation.

Sanchez and White (2011) state that "a consensus has emerged among researchers that MSER has all of the properties most desired in a truncation criterion. It is effective and efficient at mitigating bias, robust across alternate forms of biasing functions, computationally trivial, easily understood, and does not require experimenter intervention to establish parameters." Therefore, in this study, we intend to compare the proposed machine learning based approaches with MSER and MSER-5.

2.3 Machine Learning and Simulation

Machine Learning is an application of artificial intelligence, which uses mathematical models to learn without any instructions. Machine learning models are frequently used to identify patterns, extract insights, and make predictions or take actions for a given dataset. Image recognition, sentiment analysis, autonomous vehicles, product recommendations in the online markets, credit risk analysis, and language translation are just a few of the areas where machine learning methods are used. In order to achieve this, a machine learning model is trained by giving historical (and, in case of supervised learning, labeled) data to the model. Thus, the trained machine learning model becomes able to make predictions on data it has not seen before.

Simulation, on the other hand, is a fundamental modeling tool that involves computer-based models to simulate real-world events or processes. It is used to analyze and understand complex systems, evaluate the performance of the modeled system, and then design the system accordingly in a more efficient and effective way. Moreover, simulation modeling allows us to experiment with different scenarios to observe how the system behaves under changing conditions.

Machine learning and simulation, which are the most essential modeling tools in their respective fields, have started to be used together, as combining them makes it possible to solve some well-known problems.

Giabbanelli (2019) worked on machine learning and simulation integration to over-

come three different problems using machine learning, namely calibrating a simulation model, managing the experimental design of simulation modeling, and visualizing the simulation model output. In his work, it is mentioned that these two tools can be used together in the same research; however, their purposes are different. While simulation models are used to analyze the 'what-if' conditions, machine learning models are used to predict the future as the conditions stay unchanged. To express the similarities between these two concepts, Giabbanelli (2019) also states: "At a high level, these approaches proceed in very similar ways: they derive a model from some of the evidence (the 'training set' of machine learning or the 'calibration step' of simulation model) and use the remainder to evaluate the model (the 'testing set' of machine learning or the 'validation step' of simulation model)."

Laidler et al. (2020) adopted a simulation analytics perspective to analyze the output and utilized machine learning techniques to extract findings from the dynamic sample path by suggesting a k-nearest neighbor (kNN) with metric learning combination-based method.

In the research of Jain et al. (2018), two different machine learning methods, namely neural networks and Gaussian Process Regression, were studied to predict delivery dates for orders incoming to a manufacturing system. The manufacturing system simulation model is run for different scenarios, and the results gathered in a dataset are used as input to the machine learning methods.

Liu et al. (2020) stated that "The quality of large-scale logistics network simulation highly depends on the estimation of its key input parameters, which are usually influenced by various factors that are difficult to obtain." For this problem, they formulated a supervised machine learning framework to establish a relationship between the influential factors and the data. Moreover, unsupervised machine learning techniques are tested to uncover the data pattern of dynamic factors. Li et al. (2018) gathered the necessary data for the deep reinforcement learning method they worked on to solve the task selection problem of autonomous material handling vehicles by simulating the system for different scenarios.

Sherzer et al. (2022) examined whether a machine learning model can help overcome a general problem related to the queueing theory. To investigate this, they utilized

a deep learning approach to predict the steady-state queue-length distribution of an M/G/1 queueing system. Hijry and Olawoyin (2022) also modeled a queueing system and used a deep learning network to predict the waiting time in queue. Then, they compared four different optimization algorithms for learning, which are SGD, Adam, RMSprop, and AdaGrad in terms of the mean absolute error.

In the literature, there are also some studies on predicting simulation output by using machine learning methods. In order to estimate the outputs of the M/M/1 model, Sundari and Palaniammal (2015) constructed an artificial neural network that predicts the results using the parameters of the M/M/1 model without actually simulating the queueing system. Arrival rate, service rate, the number of customers in the system exceeding the queue capacity, time, and population size are used as features to predict 14 different general queueing system performance measures such as the average number of customers in system, the average number of customers in queue, and the average waiting time in system. In another similar study, Kyritsis and Deriaz (2019) estimated the average waiting time in queue for a banking system using an artificial neural network.

To the best of our knowledge, except for Lee and Kyung (1997), the machine learning based methodology of this thesis is an understudied field for the truncation point estimation problem. We could not find any other studies in the literature attempting to use machine learning methods for estimating the truncation point. In the study of Lee and Kyung (1997), the Euclidean distance method and the multilayer perceptron model were compared using M/M/1 and M/M/2 data. The M/M/1 and M/M/2 model data with different traffic intensities were given to the multilayer perceptron model as ten data points at each iteration. The multilayer perceptron model classified these data with values of 0 or 1. When a neural network categorizes data as 1, it signifies that the training data's steady-state data pattern aligns closely with the current input pattern within a specific margin of error. When the value of 1 is obtained from the multilayer perceptron network five iterations in a row, it is decided that the simulation has reached its steady-state and the truncation point is decided.

2.4 Prediction Applications of the Machine Learning Methods Used

In order for steady-state simulations to be analyzed accurately, the transient phase of the simulation, which represents the duration before it reaches the steady-state, should not be included in the analysis. There are many methods to determine this truncation point value, as mentioned in Section 2.2. In this study, based on the increasing use of simulation and machine learning concepts in a cooperative manner, we intend to explore if some of the machine learning techniques can also be used in estimating the truncation point. In this section, we discuss in which areas and for what purposes the three candidate machine learning methods, namely Multilayer Perceptron (MLP), Long Short-Term Memory (LSTM), and Conditional Recurrent Neural Network (CRNN) are used for prediction in the literature.

Deshpande (2012) introduced a MLP network as a solution option for forecasting rainfall time series. The outcomes of the network demonstrate that the MLP network performs favorably in predicting rainfall time series compared to another neural network based on mean square error and normalized mean square error.

Kumar and Jha (2013) employed a MLP network to forecast two significant weather parameters: maximum and minimum temperatures. The network was trained and tested using actual historical data to predict these temperature values. They examined the network's performance according to the mean square error function and stated that the MLP network holds promise for effective weather forecasting applications.

Derbentsev et al. (2020) addressed the challenges associated with short-term financial time series prediction through machine learning techniques such as support vector machine, MLP, random forests, and stochastic gradient boosting Machine. The dataset consisted of daily closing prices from two stock indices, two well-known cryptocurrencies, and an exchange rate. Among these machine learning approaches, the MLP network stands out as one of the optimal choices based on its mean absolute percentage error value obtained from historical price information.

Yulita et al. (2021) worked on forecasting the number of COVID-19 cases for the next 30 days by utilizing multilayer perception and linear regression. The dataset contains positive COVID-19 cases collected for five months. They concluded that the

MLP network's performance is better than the linear regression with respect to the root mean squared error values.

In the literature, there are some studies where both MLP and LSTM methods have been used and compared for various purposes.

Nelson et al. (2017) aimed to forecast the forthcoming patterns in stock prices by utilizing historical price data with technical analysis indicators. To accomplish that, an LSTM network is constructed. With only minor exceptions, the LSTM network exhibits superior performance, and the results of the LSTM network are highly encouraging, as the network's prediction capabilities stand out in comparison to those of the MLP, random forest, and pseudo-random model.

Predescu et al. (2019) stated that "Software effort estimation is the biggest challenge for project managers is to meet their goals within the given time limit." As a result, they focused on demonstrating feasibility of employing neural network algorithms for this problem by working with MLP and LSTM. With 77 sample project data, the MLP outperformed the LSTM in software effort estimation by resulting in a superior determination coefficient.

When the studies with only LSTM are examined, they can be found in many different domains. Some examples are as follows. Houdt et al. (2020) provided both insights into the LSTM network and shared his findings from the literature concerning the domains where the LSTM network finds application. As in Houdt et al. (2020), the utilization of LSTM networks spans various domains and objectives. During our literature review, our attention was directed toward research centered around time series prediction, aligning with the scope of our study.

Fischer and Krauss (2018) employed LSTM network for the prediction of a financial market index. Their analysis includes a comparison between LSTM, random forest, conventional deep neural network, and basic logistic regression models. Remarkably, LSTM, which aligns well with this specific field, exhibits a significant performance advantage over both the standard deep neural network and logistic regression methods.

In the study by Sagheer and Kotb (2019), a version of the LSTM network, with mul-

tiple LSTM layers, is utilized to solve the time series forecasting problem of the petroleum industry. They showed that when the data length increases, the LSTM network performs better to understand the connections in the dataset than the artificial neural network.

Xuan-Hien et al. (2019) worked on a flood prediction model utilizing a LSTM neural network on daily discharge and rainfall data. The LSTM model effectively captures relationships among data and shows accuracy with its predictions.

Nguyen et al. (2020) addressed two essential problems in supply chain management: sales prediction and anomaly detection in sales. However, for our context, we will only mention predicting retail sales. The LSTM network for multivariate time series has been presented for this purpose, and the obtained LSTM results have shown that the LSTM network worked well for the purpose of future sales predictions considering RMSE.

There are some additional studies involving use of the CRNN. Remy (2020) utilized the CRNN network to predict the weather temperature of a particular city. The weather behaves differently depending on the city, therefore it may be useful to condition temperature prediction on the city. Kotsias et al. (2019) studied the CRNN and molecule side information. Descriptor conditions representing molecules are also given as input to the CRNN. By doing this, the CRNN used offered better results by focusing on a specific protein target. Inspired by this work, Mohapatra and Gómez-Bombarelli (2020) compared CRNN and a graph-based genetic algorithm in the domain of molecular optimization and concluded that the CRNN barely performed better.

To summarize the review in this section, artificial neural networks such as MLP have been used in different studies for estimating the outputs of queueing models, but it has not been used for truncation point estimation. On the other hand, no study was found making use of LSTM or CRNN in this field. However, these three machine learning methods have been widely used in literature for time series prediction or forecasting purposes. Hence, in this study, we intend to explore potential use of these three machine learning methods for estimating the truncation point in steady-state simulations.

CHAPTER 3

PROBLEM DEFINITION AND BACKGROUND

In this chapter, first, the simulation initial bias problem studied in this thesis will be defined in Section 3.1. Then, the autoregressive model and the M/M/1 model, which are frequently used as representative models in studies on this problem, will be explained in Sections 3.2 and 3.3, respectively.

3.1 The Initialization Bias Problem

Simulation modeling is an essential method used for solving real-life problems. Simulation models are used in many different areas, such as highway and street traffic models, natural disaster models, cost models, combat models, and queueing system models. Thanks to simulation models, it is possible to predict what may happen in complex systems or to analyze the performance of a new system to be installed, and the system can be made more efficient and effective in line with these results. However, there are various problems to be handled in conducting a simulation study, such as model oversimplification problem, input data uncertainty problem, model calibration and validation problem, and statistical output analysis problems. In this study, the main concern is the initialization bias problem, which is faced in analyzing the output of stochastic steady-state simulations.

According to Banks et al. (2005), steady-state simulation is defined as "simulating the system until it reaches a stable operating condition where the behavior of the system variables exhibits no significant changes over time". Most simulation models start with an initial state where the system is idle and empty, but the steady-state characteristics to be examined are generally not similar to the initial state. According to Law

(2015), the transient state in a simulation is defined as "... the period of time from the beginning of the simulation until the system variables reach a steady pattern of behavior. During this time, the variables may exhibit erratic or inconsistent behavior, making it difficult to draw meaningful conclusions about the system's performance.". Since the values in the transient phase are directly affected by the initial state and do not represent the steady-state, the analysis to be made on the simulation model outputs will not be accurate and healthy. Therefore, the observations encountered in the transient phase must be eliminated in order to analyze the simulation model output. In the simulation literature, this problem is called the initialization bias problem.

To explain this problem more clearly, let us consider a queueing model simulation. Assume that the expected time spent in the system in the steady-state of this queueing model is equal to a relatively high but unknown value and that the simulation model is initially idle and empty. As the simulation progresses, values of the individual time in system observations will increase gradually because the simulation entities (for example, customers) that will come at the very beginning will only spend a short time in the system since the system is initially empty and idle. In order for simulation entities to accumulate in the system, a specific time must pass, and after a certain time, the simulation will reach its steady-state provided that the overall traffic intensity is less than one. Including the initial observations in the analysis will artificially reduce the average time spent in the system and cause misleading results by underestimating the expected value. In the opposite scenario, the same problem applies to a simulation model that starts with an overcrowded and busy system. The only difference is that, in this case, there will be large initial observation values that will increase the steady-state average and cause overestimation of the true mean. In order to overcome this problem and to obtain more accurate results, the point before which the observations are to be excluded from the analysis should be found as the truncation point.

In general, simulation modeling is used to study complex real-life systems for which neither the true steady-state output value (expectation) nor the truncation point are known. Therefore, in literature, studies dealing with the initial bias problem make use of some typical stochastic systems for which the steady-state expected value of the output is known. Output data generated from simulation models of such stochastic systems are used as a proxy representing the output of simulations of real-life systems.

Output data obtained in this manner are then used in developing and testing methods for detecting the truncation point. In testing the proposed method, the steady-state expected value of the stochastic system output is estimated by using the data that come after the detected truncation point. Then, this estimate is compared with the known expectation to see if it is sufficiently close to the true value.

When the studies in literature on the initialization bias problem are examined, two different proxy models become prominent. These are the autoregressive model and the M/M/1 queueing system model. For this reason, the output data from these two models were used as datasets in this thesis study as well. More detailed information about these models are given in Sections 3.2 and 3.3.

3.2 Autoregressive (AR) Model

Autoregressive models are widely used in the analysis of natural phenomena, economics, and other time-varying processes due to their ability to capture the influence of past values on current values. In addition, they have recently been used for neural network prediction models since the autoregressive models are incredibly flexible and can model many different types of time series patterns. In different researches such as Fishman (1978), Schriber and Andrews (1984), and Bischak et al. (1993), autoregressive processes have been utilized in simulation research to capture the autocorrelation characteristics of output processes. Wang and Wong (2002) propose a model-based technique for detecting and diagnosing gear faults. The proposed technique establishes an autoregressive model on the vibration signal of the gear of interest in its healthy state. The model is then used as a linear prediction error filter to process the future-state signal from the gear.

Autoregressive models are briefly shown as AR(p) where the parameter p represents the order of the model. For instance, the AR(1) autoregressive process is a process in which the current value depends only on the previous value, whereas the AR(2) process is the process in which the current value depends on the two previous values. In the context of autoregressive modeling, time series are computed by considering the correlation between preceding and subsequent data points. Therefore, this method-

ology can be regarded as a statistical technique. The mathematical representation of the AR(p) model can be seen in Equation 3.1.

$$X_i = \phi_1 X_{i-1} + \phi_2 X_{i-2} + \phi_3 X_{i-3} + ... + \phi_p X_{i-p} + \varepsilon_i \tag{3.1}$$

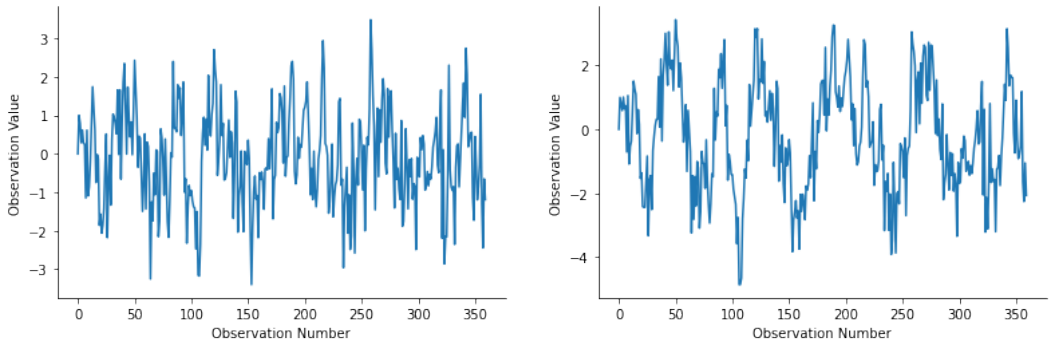
where X_i is the i^{th} observation in the time series, ϕ_i is the coefficient of respective observation, and ε_i is the error term.

As can be seen from the equation, as the coefficients in an autoregressive model take different values, it is possible to generate data that represent very different patterns, and the error term represents the stochastic component. In this thesis study, typically the 2^{nd} order autoregressive model data are used. Generation of two different sample datasets with a length of 360 are given in Equations 3.2 and 3.3, and generated time series are visualized in Figure 3.1.

$$X_i = 0.5X_{i-1} + \varepsilon_i \tag{3.2}$$

$$X_i = 0.5X_{i-1} + 0.3X_{i-2} + \varepsilon_i \tag{3.3}$$

where ε_i is distributed as $N\,(0, 1)$ for both equations.



(a) 1^{st} Order AR Model Data

(b) 2^{nd} Order AR Model Data

Figure 3.1: Examples of Visualization for AR Model Data

Other datasets can be obtained by adding different bias types encountered in real life to the autoregressive model data. Example time series with different types of bias, which are used in this work (more detailed information will be given in Section 5.1),

can be found in Equations 3.4 through 3.9 and Figure 3.2.

$$X_i = 0.5X_{i-1} + \varepsilon_i + 5e^{-0.005(i-1)} \tag{3.4}$$

$$X_i = 0.5X_{i-1} + 0.3X_{i-2} + \varepsilon_i + 5e^{-0.005(i-1)} \tag{3.5}$$

$$X_i = 0.5X_{i-1} + \varepsilon_i + 5/5 \tag{3.6}$$

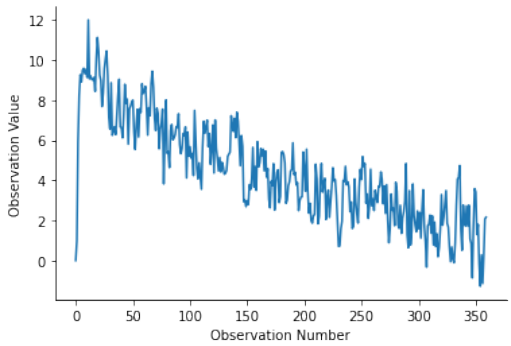
$$X_i = 0.5X_{i-1} + 0.3X_{i-2} + \varepsilon_i + 5/5 \tag{3.7}$$

$$X_i = 0.5X_{i-1} + \varepsilon_i + 5e^{-0.005(i-1)} \sin\left(\frac{i\pi}{200} + \frac{\pi}{2}\right) \tag{3.8}$$

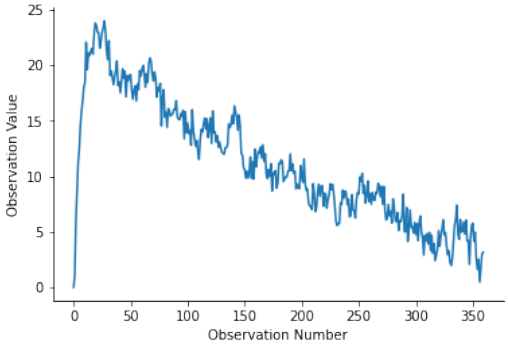
$$X_i = 0.5X_{i-1} + 0.3X_{i-2} + \varepsilon_i + 5e^{-0.005(i-1)} \sin\left(\frac{i\pi}{200} + \frac{\pi}{2}\right) \tag{3.9}$$

where ε_i is distributed as $N(0, 1)$ for the equations.

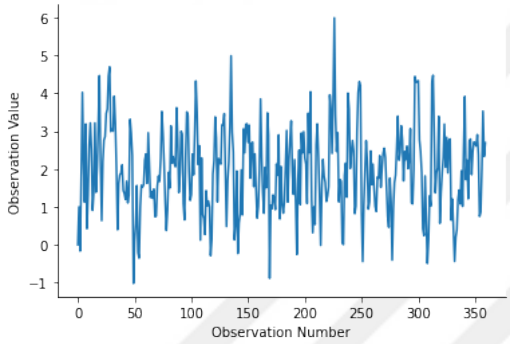
Due to this diversity advantage, the autoregressive model has been used in many studies in literature concerning simulation output analysis as mentioned above. It has also been utilized in studies on truncation point estimation. Hoad et al. (2008) studied warm-up length estimation with the autoregressive model. While doing this, they used different coefficient values of the autoregressive model. They worked on first, second, and fourth-order autoregressive models and used error values with varying distributions for these models. White et al. (2000) compared five different truncation point heuristics using the autoregressive model. In making this comparison, the study is conducted by using six different coefficient values for the AR model, and three different bias functions are added to the AR model data. In addition, the ease of implementation is another advantage of the autoregressive model. Due to these advantages, a variety of autoregressive models are also used in this thesis.



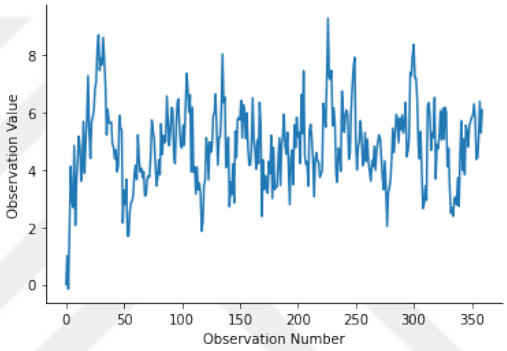
(a) AR(1) with Exponential Bias



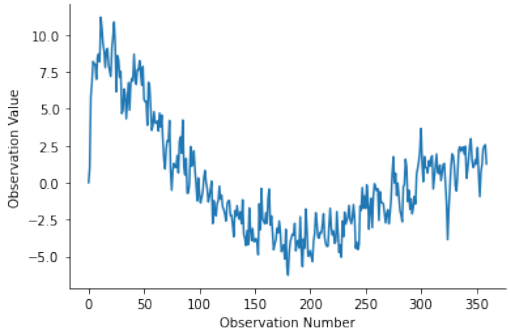
(b) AR(2) with Exponential Bias



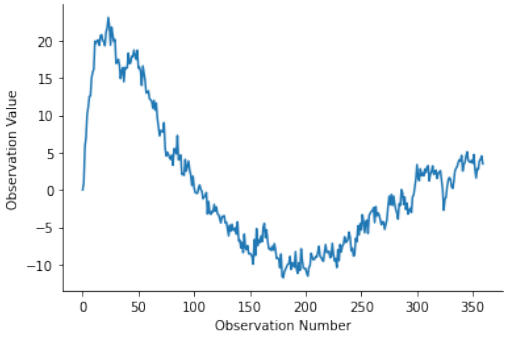
(c) AR(1) with Mean Shift Bias



(d) AR(2) with Mean Shift Bias



(e) AR(1) with Oscillation Bias



(f) AR(2) with Oscillation Bias

Figure 3.2: Examples of Visualization for AR Model Data with Different Types of Bias

3.3 M/M/1 Queueing System Model

The M/M/1 is a queueing system model used to understand and analyze the behavior of waiting lines. It is an analytical model representing a single-server queue with Poisson arrivals and exponential service times. The M/M/1 model is generally used in various areas such as transportation, telecommunications, and manufacturing to analyze the system in terms of selected criteria. The model helps to gain an understanding of different performance measures, such as the average number of customers in queue or system, and the average waiting time in queue or system. According to such M/M/1 model results, the system can be modified and optimized.

The M/M/1 model has been used in many different studies. In his well-known textbook, Law (2015) extensively discussed results of various experiments with M/M/1 simulation concerning both elimination of initialization bias and statistical output analysis for estimation of performance measures. Afolalu et al. (2019) worked with many different models for the banking sector including the M/M/1 model and made suggestions to improve productivity performance. Barroso (2018) worked on M/M/1 data to analyze the effectiveness of several methods in eliminating initial bias from steady-state stochastic simulations. Modi et al. (2019) established a M/M/1 queueing model and conducted a thorough analysis of the queueing theory to evaluate the traffic intensity at the Palasia intersection in Indore city, India, to examine the optimal lane configuration and signal timing parameters with a certain level of precision. Keshtgary et al. (2012) proposed an analytical model for estimating energy consumption in cluster-based underwater wireless sensor networks using the M/M/1 queueing model. This model is used to examine the network performance in terms of average energy consumption. On the initialization bias problem, Law (2015) tested different bias elimination techniques using M/M/1 data. Moreover, Hyung Sool Oh and Kyoung Jong Park (2015) compared MSER-5 and exponential variation rate methods using the M/M/1 model with four different traffic intensities in their study.

In order to better understand the M/M/1 model, it is helpful to examine what these abbreviations mean. Kendall (1953) included basic definitions in his work, and the essential notation for this queueing model are given below.

Generally, a queueing model is represented by A / S / c / k / n / d where

- A: Inter-arrival time distribution
- S: Service time distribution
- c: Number of servers available
- k: Waiting line capacity (default = ∞)
- n: Customer population size (default = ∞)
- d: Scheduling discipline (default = First-In-First-Out or FIFO).

There are three different options representing the interarrival and service time distributions. These are:

- D: Deterministic (constant)
- M: Markovian/Memoryless (exponential distribution)
- G: General/arbitrary distribution (possibly with known mean and variance).

To summarize, based on Kendall's notation, the M/M/1 model has exponential interarrival and service time distributions, a single server, unlimited capacity for entities waiting in queue, and it works with the FIFO discipline. A graphical representation is given in Figure 3.3.

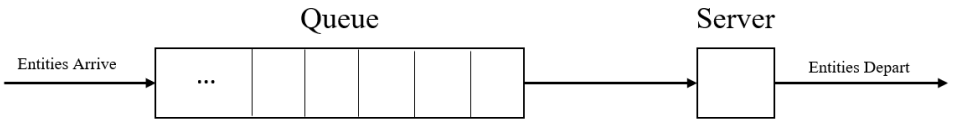


Figure 3.3: Graphical Representation of the M/M/1 Model

In queueing theory, arrival and service rates are two fundamental parameters that are vital in determining the system's behavior. The arrival rate (λ) is defined as the average number of arrivals per unit time, while the service rate (μ) is defined as the

average number of customers served per unit time. Poisson process and the exponential distribution are used to model the arrival rate and service rate to predict the behavior of the model and optimize the system's performance. In order for the M/M/1 model to reach steady-state, the ratio of the arrival rate to service rate, which is the traffic intensity (ρ), must be less than one.

$$\rho = \frac{\lambda}{\mu} < 1 \tag{3.10}$$

If the condition in Equation 3.10 is not met, the M/M/1 model cannot reach steady-state because incoming entities start to accumulate in queue and, since the queue capacity is infinite in the M/M/1 model, the number of entities approaches infinity. The traffic intensity in queueing theory is often referred to as the utilization of a server in simulation. A high traffic intensity means that the system is heavily utilized, while a low traffic intensity indicates that the system is underutilized. Moreover, as the utilization increases, the time for the simulation model to reach the steady-state also increases. This makes the truncation point estimation more difficult.

The Little's Law is a well-known concept in queueing theory that enables finding theoretical values of performance measures such as expected values of the number of entities in queue and in system, and the time spent in queue and in system. Therefore, it is widely used for analyzing the performance of various queueing models. Little (1961) states that the average number of entities in a system can be found by Equation 3.11.

$$L = \lambda W \tag{3.11}$$

where L is the average number of entities in a queueing system, λ is the average number of entities arriving at the system per unit time and W is the average waiting time an entity spends in system. Kleinrock (1975) states the mean waiting time in system as in Equation 3.12.

$$W = \frac{1}{\mu - \lambda} \tag{3.12}$$

Expected waiting time in queue can be calculated by subtracting the mean service time as in Equation 3.13.

$$W_q = \frac{1}{\mu - \lambda} - \frac{1}{\mu} = \frac{\lambda}{\mu\,(\mu - \lambda)} \tag{3.13}$$

Expected values of the number of entities in queue and in system can also be derived from these equations. In testing our truncation point estimation method, steady-state output estimates found using the data that comes after the detected truncation point of the M/M/1 simulation model can be compared with these theoretical values.

In implementing the M/M/1 simulation model, the recursive relation given in Equation 3.14 can be used to calculate the waiting time (or delay) of entities in queue.

$$X_i = \max\{0, C_{i-1} - t_i\} \tag{3.14}$$

where

- A_i is the interarrival time between entities *i* and $i - 1$
- $t_i = t_{i-1} + A_i$ is the arrival time of entity *i* ($t_1 = 0$),
- S_i is the service time of entity *i*,
- X_i is the delay in queue of entity i ($X_1 = 0$), and
- $C_i = t_i + X_i + S_i$ is the service completion time for entity *i*

We implement the M/M/1 simulation model in this study using Equation 3.14. v

CHAPTER 4

SOLUTION APPROACHES

In this chapter, we describe the machine learning based solution approaches we propose in order to estimate the truncation point in an attempt to solve the initialization bias problem in steady-state simulations. We employ three different types of artificial neural networks, namely Multilayer Perceptron Regressor (MLPR), Long Short-Term Memory (LSTM), and Conditional Recurrent Neural Network (CRNN). In the three sections below, first a general definition (structure and working principles) of each network type is provided, and then how the network is used in this study is explained.

4.1 Multilayer Perceptron Regressor (MLPR)

4.1.1 General MLPR Definition

The MLPR network is a version of Multilayer Perceptron (MLP) constructed by modifying the output layer so that the network can predict continuous target variables by performing regression tasks. Therefore, to explain the MLPR network, we focus on the MLP network.

In order to better understand the MLP network, it is necessary to start with the perceptron first. The perceptron is the part of the MLP network that represents the learning capability. The perceptron transmits the output according to an activation function after multiplying the input values transmitted to it by their weights. The working principle of Perceptron is shown in Figure 4.1 as also schematized by Olmedo et al. (2018).

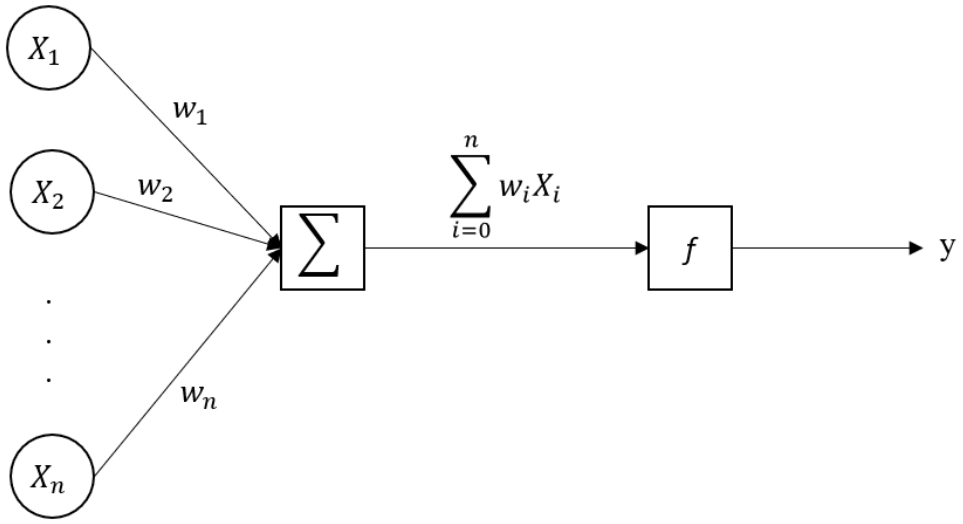


Figure 4.1: Principle of Perceptron (Olmedo et al., 2018)

In Figure 4.1, $X_1, ..., X_n$ represent the input, which are values of features of a sample to be used in training the network. $w_1, ..., w_n$ represent the respective weights for the feature values, f is the activation function of the perceptron, and y is the output.

Perceptron can solve linearly separable functions, as can be seen from the weighted sum formula. However, the perceptron cannot be successful when faced with scenarios such as the *exclusive or (XOR)*problem.

As displayed in Figure 4.2, the XOR problem is a non-linear problem. In other words, one cannot divide the 1's and 0's with a line. Singh and Pandey (2016) state that "It is not possible to solve the XOR problem using the single layer network because of presence of nonlinearity in the problem exhibited by XOR logic." As a result, multiple layers are required to solve the XOR problem. For example, two perceptrons that have learned the *not and* and *or* operators in the first layer and a neural network with the "and" operator in the other layer have the ability to solve the XOR problem. A visualization of this example can be seen in Figure 4.3.

Therefore, the multilayer perceptron network can be considered to solve non-linear problems.

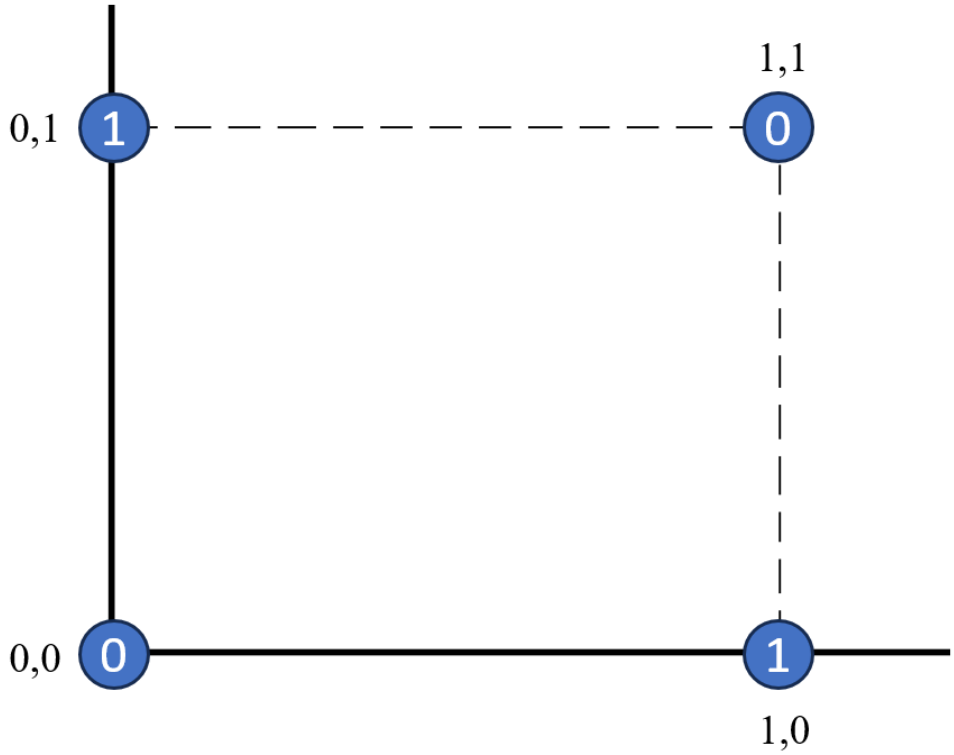


Figure 4.2: Graphical Representation of the XOR Problem

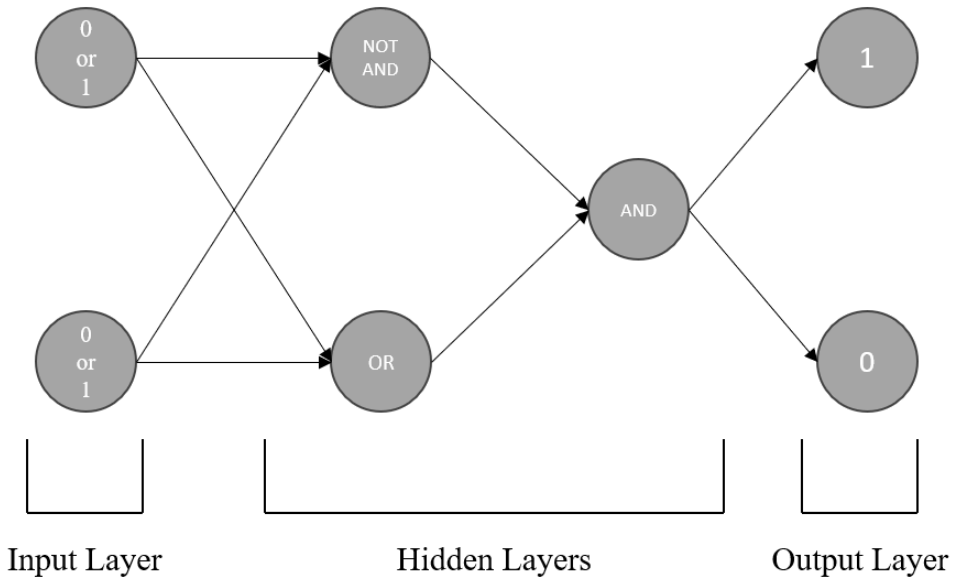


Figure 4.3: Example Solution for the XOR Problem

The MLP is a well known and frequently used neural network type. It has multiple layers of interconnected nodes, with each node being a simple processing unit that receives input from other nodes and produces output. In the MLP network, information flows in only one direction, that is, from input to output without any loops. In other words, MLP is a fully connected feed-forward neural network. It is widely used in machine learning applications, particularly for supervised learning tasks such as classification and regression. An MLP network possesses at least three layers, namely an input layer, a hidden layer, and an output layer. MLP earns its capabilities by incorporating multiple hidden layers having numerous nodes. Each node within a hidden layer receives its input from the previous layer and generates its output value using an activation function. The output from a hidden layer is propagated to the subsequent layer until it reaches the output layer, where the final output is obtained. The learning process of the MLP involves the following steps:

- Forward propagate data through the network from the input layer to the output layer.
- Compute the error, which is the difference between the predicted output and the actual output.
- In order to minimize the error calculated in the earlier step, backpropagate the error while taking its derivative with respect to each weight in the network, and then update the weights accordingly.
- Iterate through the first three steps to learn the optimal weights for the MLP.

Detailed explanation of the above steps can be found in Noriega (2005).

MLPR refers explicitly to an MLP network designed for regression tasks, where the output to be predicted is continuous rather than categorical. Both MLP and MLPR networks are based on the same network architecture. In other words, MLPR can be considered as a subset of MLP as the output layer of a MLPR has an activation function such that the network can produce continuous numeric values for predictions, while the activation function used in the MLP can be chosen according to the task, which is typically classification.

4.1.2 Proposed MLPR Configuration

Artificial neural networks have some essential properties and hyperparameters that determine the neural network's architecture. Changing these properties and parameter values may significantly affect the performance of the network in solving a particular problem. Therefore, their selection is as critical as the selection of the network type.

In case of MLPR, the number of hidden layers, the number of nodes in each hidden layer, the activation, loss and solver functions to be used, and the maximum number of training iterations need to be selected to come up with the proper network configuration to be able to effectively and efficiently solve the problem at hand.

A comprehensive hyperparameter tuning experiment has been carried out for the methods proposed as a solution approach in this study. Details of this experiment and performance of the network with a variety of configurations tried are given in Section 5.4. The configuration of the MLPR network used for estimating the truncation point is determined based on the results of this experiment and summarized below.

As shown in Figure 4.1, a node found in the hidden layer transmits the result obtained from the activation function, together with the incoming values and the weights of these values, to the next node. Therefore, the number of hidden layers and the number of nodes in each layer directly affect the performance of the MLPR network, as they represent the way to predict the input values given to the network. In this study, we propose that the MLPR network should have three hidden layers, and have 200, 150, and 100 nodes in these hidden layers, respectively, in order to make truncation point estimation more successful. Graphical representation of the proposed network architecture is given in Figure 4.4.

Note that nodes in the input layer represent the observation values (a time series) obtained from a simulation replication, and the single output node represents the estimated truncation point. The reason why the input layer has 1000 nodes in Figure 4.4 is that the length of the time series data in the experiments (simulation replication length), to be explained in Chapter 5, is equal to 1000.

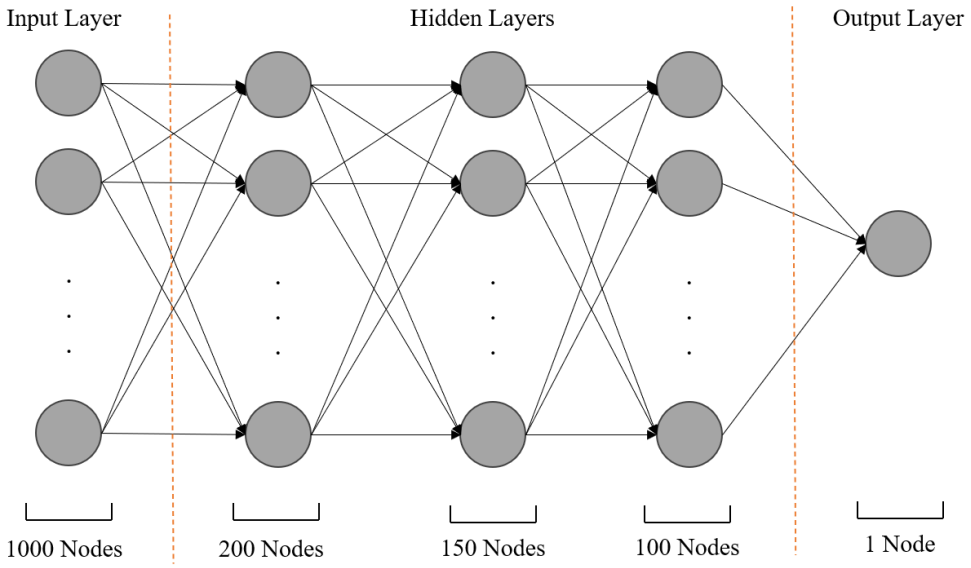


Figure 4.4: Layers of the Proposed MLPR network

As the activation function, we used the ReLU function. In a neural network, the activation function is crucial in transforming the weighted sum of the input values of a node into its output. The rectified linear activation function, known as ReLU, is a piecewise linear function that directly outputs the input with a basic algorithm. If the input of ReLU is negative or zero, the function returns the value of zero. Otherwise, the function gives the input value as the output (Agarap, 2018). ReLU has gained popularity as the default activation function in many neural networks, including MLP, due to its ease of use in training and ability to achieve improved performance (Zeiler et al., 2013). The formulation and a graphical representation of ReLU activation function is given in Equation 4.1 and Figure 4.5.

$$y = \max(0, x) \tag{4.1}$$

where x represents the input value (that is, weighted sum of the input values of a node) and y represents the output of the node.

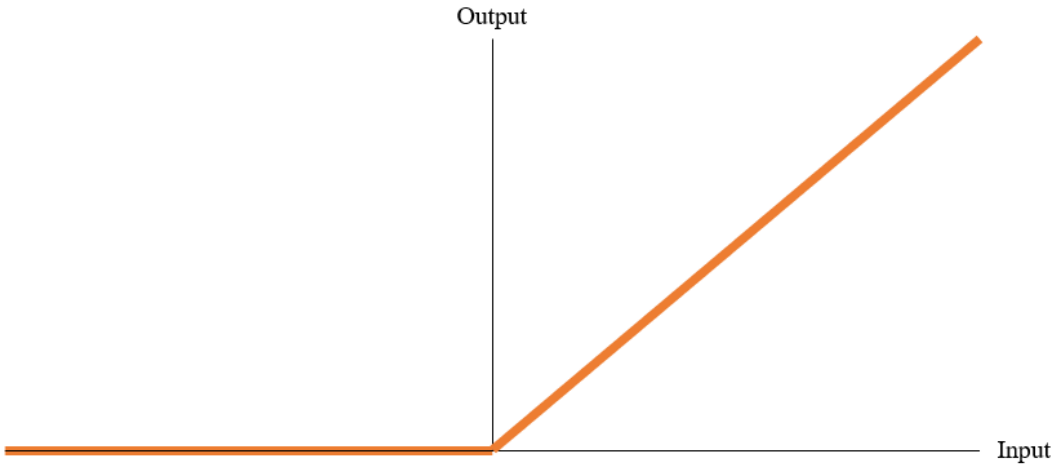


Figure 4.5: Graphical Representation of the ReLU activation function

The loss function in machine learning represents closeness of the machine learning network's predictions to the actual target values. The primary purpose of training a machine learning network is to update the weights and bias values so that the loss function value is minimized. Neural network estimations become more successful as the loss function value decreases. Since we are working with the MLPR network and actually working on a regression task, we set the loss function as mean squared error (MSE). MSE is a loss function that is used in regression tasks to calculate the average squared difference between the predicted target variable values and the actual target variable values. Formulation of MSE can be found in Equation 4.2 (James et al., 2013).

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2 \tag{4.2}$$

where n represents the sample size (number of samples in the training dataset), y_j is the actual target value and $\hat{y}_j$ is the predicted target value for the j^{th} sample.

In our work, Adam was selected as the solver function. Adam is an alternative optimization algorithm that replaces the conventional stochastic gradient descent technique to update network weights while training the network iteratively. Adaptive Moment Estimation, known as Adam, is a highly efficient optimization algorithm for gradient descent. Adam distinguishes itself when handling large-scale problems in-

volving extensive amounts of data or parameters, and it consumes less memory while maintaining effectiveness. Essentially, Adam combines the principles of the gradient descent with momentum algorithm and the Root Mean Square Propagation (RMSP) algorithm. (Kingma and Ba, 2014)

Lastly, the maximum number of iterations represents the stopping condition for the solver. The solver will continue iterating until either convergence or the specified number of iterations is reached. When Adam is used as the solver, the maximum number of iterations determines how many times each data point can be used during the training process, rather than the number of individual gradient steps. After trying some larger values, we set the maximum number of iterations to 200 for the MLPR network, as the network stabilized at this point.

To summarize, the MLPR architecture and parameter values are selected as listed below.

- Hidden layer sizes: 200, 150, 100
- Activation function: ReLU
- Loss function: MSE
- Solver function: Adam
- Maximum number of iterations: 200

4.2 Long Short-Term Memory Network (LSTM)

4.2.1 General LSTM Definition

LSTM is a special case of the recurrent neural networks. Hochreiter et al. (1997) state that "Recurrent networks can use their feedback connections to store representations of recent input events in the form of activations. The most widely used algorithms for learning what to put in short-term memory, however, take too much time to be feasible or do not work well at all, especially when minimal time lags between inputs and corresponding teacher signals are long. Although theoretically fascinating,

they do not provide clear practical advantages over backpropagating in feed forward networks with limited time windows." LSTM is introduced to overcome this problem (Hochreiter et al., 1997).

A LSTM unit consists of four components: a cell, an input gate, an output gate, and a forget gate. The input gate is responsible for updating the cell state considering the previous hidden state and the current input. The output gate controls the information that will be passed through as the output with respect to the previous hidden state and current input. Initially, the forget gate was not included in the LSTM network; however, Gers et al. (2000) proposed its addition to enable the network to reset its state. The cell retains information over flexible time intervals, while the three gates control the information flow within the cell. The forget gate is responsible for selecting whether information is retained or forgotten according to the output of an activation function. A typical LSTM unit can be shown as in Figure 4.6.

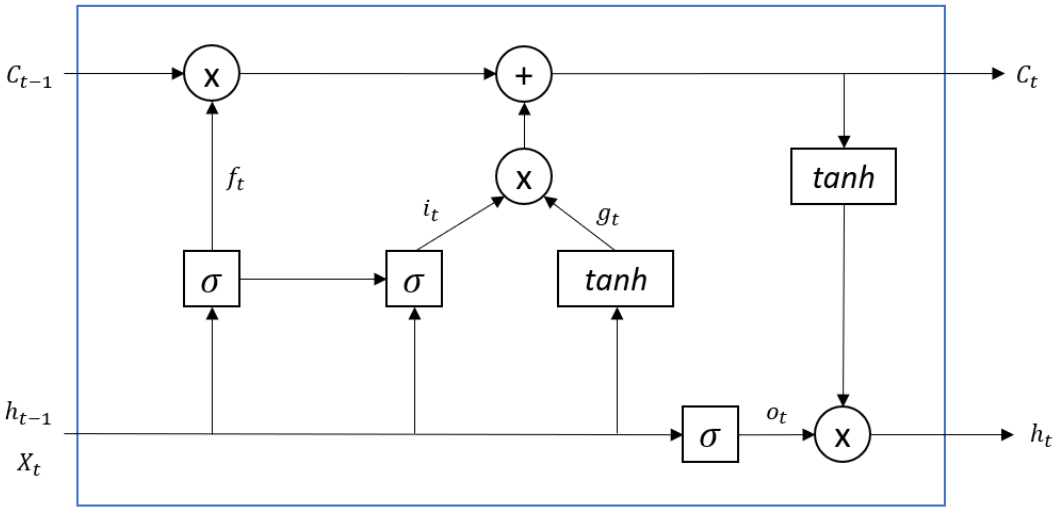


Figure 4.6: Example Representation of LSTM Unit

Figure 4.6 also provides information on how an LSTM unit works as we follow the directions. This is explained below, following the notation.

- X_t: Input at time step t
- h_t: Hidden state at time step t

- C_t: Cell state at time step t
- b_f, b_g, b_i, b_o: Bias vectors
- $W_{f,x}$, $W_{g,x}$, $W_{i,x}$, $W_{o,x}$, $W_{f,h}$, $W_{g,h}$, $W_{i,h}$, $W_{o,h}$: Weight matrices
- f_t, i_t, g_t, o_t: Activation function value vectors
- σ and $tanh$: Sigmoid and hyperbolic tangent activation functions

According to Figure 4.6, the calculations within a LSTM unit are given in Equations 4.2 through 4.7.

$$f_t = \sigma\left(W_{f,x}X_t + W_{f,h}h_{t-1} + b_f\right) \tag{4.3}$$

$$g_t = tanh\left(W_{g,x}X_t + W_{g,h}h_{t-1} + b_g\right) \tag{4.4}$$

$$i_t = \sigma\left(W_{i,x}X_t + W_{i,h}h_{t-1} + b_i\right) \tag{4.5}$$

$$o_t = \sigma\left(W_{o,x}X_t + W_{o,h}h_{t-1} + b_o\right) \tag{4.6}$$

$$C_t = f_t \times C_{t-1} + i_t \times g_t \tag{4.7}$$

$$h_t = o_t \times \tanh\left(C_t\right) \tag{4.8}$$

where $\times$ is the dot product.

A LSTM unit generates its output C_t and h_t according to these calculations. The weights in the equations are updated by backpropagation.

LSTMs have complex recurrent connections that allow information to flow from one time step to the next within a sequence. The connections in MLPs are feed-forward between the layers in a straightforward manner, where there are no loops or recurrent connections. On the other hand, each gate in a LSTM network has its own set of weights as explained, which determine how the data are processed and can be remembered over time. Therefore, the LSTM network has more sophisticated weight structures than the MLP network, which allows LSTMs to capture long-term dependencies and patterns in sequential data.

4.2.2 Proposed LSTM Configuration

We experimented with different architectures of the LSTM network. In these experiments, the main parameters changed was the number of LSTM units in hidden layers of the network. We first started our studies on the LSTM network by examining how the architecture of the MLPR network, which we had successful results with, would perform with the LSTM network. When hidden layers in LSTM were set up as in MLPR, we faced both memory and computing time problems. To deal with these problems, the time series obtained from a simulation replication was shortened by batching the observations and using the batch means. Length of the time series given as input to LSTM was first reduced from 1000 to 200 by taking the average of every 5 consecutive simulation observations, and then to 100 by averaging every 10 consecutive observations, as in Equations 4.8 and 4.9, respectively.

$$Z_t = \frac{1}{5}\sum_{i=1}^{5} X_{5(t-1)+i} \quad for \quad t = 1, ..., 200 \tag{4.9}$$

$$Z_t = \frac{1}{10}\sum_{i=1}^{10} X_{10(t-1)+i} \quad for \quad t = 1, ..., 100 \tag{4.10}$$

where X_t is the t^{th} observation obtained in a simulation replication and Z_t is the input given to the neural network at time step t.

When LSTM experiments were performed with an input of length 200 time steps, the hidden layer sizes were chosen as 40, 30, and 20 to be proportional with the length of the time series. Similarly, for a time series of length 100, the LSTM network had 20, 15, and 10 nodes in the three hidden layers, respectively.

In addition to the hidden layer sizes, the number of training epochs needs to be determined for the LSTM. For this, initially the maximum number of iterations used for the MLPR was considered. However, training the LSTM network for 200 epochs was not possible due to the memory and computing time problems mentioned earlier. Therefore, experiments had to be performed with a lower number of epochs. In our study, different values were tested for the number of epochs. However, increasing this value did not change the model's performance, therefore the number of epochs was

determined as 20.

Even though the hidden layer sizes in LSTM differ for different input lengths, other parameter values remain the same as in the MLPR configuration. To summarize, the selected parameter values for the LSTM network are listed below.

- Hidden layer sizes:
  - 200, 150, 100 for input length = 1000
  - 40, 30, 20 for input length = 200
  - 20, 15, 10 for input length = 100
- Activation function: ReLU
- Loss function: MSE
- Solver function: Adam
- Number of epochs: 20

4.3 Conditional Recurrent Neural Network (CRNN)

4.3.1 CRNN Definition and Usage

In this study, the last machine learning network used for truncation point estimation is the CRNN. Remy (2020) states that "The conditional recurrent layer is useful if you have time series data with external inputs that do not depend on time." From this statement, we understand that if a neural network is also given an auxiliary time-invariant sample feature, the CRNN architecture can be used to analyze whether its performance will improve. In our case, the type of initial bias in simulation outputs that are frequently studied in real life can be known. For example, for the queueing system simulations that start empty and idle, the delay in queue starts at zero and approaches the steady-state value in an exponential manner. Therefore, in addition to the input data, the bias type can also be given to the neural network. For this, studies were carried out on the CRNN network with the AR model with three different bias types (exponential, mean shift, and oscillation) described in Section 3.2.

For recurrent neural networks (RNNs), the input is a sequence of time steps, with each time step containing a tensor of features. Therefore, it is possible to have multiple features at each time step. However, if one of these feature is not time series-based, it might not be suitable to directly pass it through the RNN since the auxiliary input, in our case the bias type, is not dependent on time. The RNN may still work but it may lead to higher loss or lower accuracy while training the network. An alternative approach is to incorporate this additional information into the RNN network by using extra layers. Then, CRNN combines the auxiliary inputs with the initial RNN outputs and continues to train the network, effectively making it a multi-input network (time series data and auxiliary inputs). To initialize the RNN states, a learned representation of the conditions can be used as a starting point. Remy (2020) visualizes this method as in Figure 4.7.

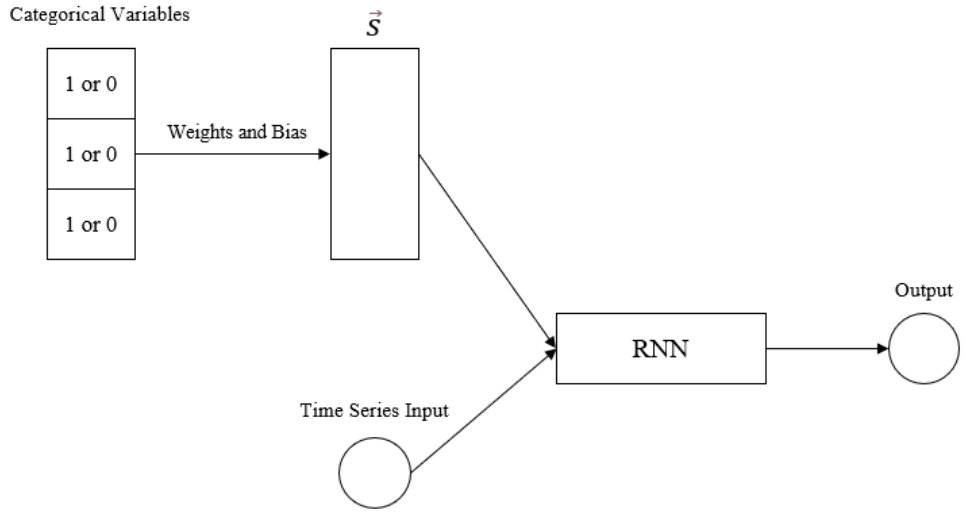


Figure 4.7: Graphical Representation of CRNN Network (Remy, 2020)

As an example, Remy (2020) utilized the CRNN network to predict the weather temperature of a particular city. The weather behaves differently depending on the city, therefore it may be useful to condition temperature prediction on the city. For this estimation, the 30 cities whose temperatures are highly correlated with temperature of the selected city are determined. As exogenous component (or auxiliary feature) Remy (2020) gave as input to the network the last day's temperature of the correlated cities. In the example, the performance has improved with respect to the mean

absolute error metric when compared to the LSTM network's performance.

4.3.2 Proposed CRNN Configuration

Discovering the common aspects of truncation point estimation with the findings of Remy (2020), we decided to adapt his solution approach for our problem. In our study, in addition to our time series data, there are bias types of the AR model that are not dependent on time. The exponential, mean shift, and oscillation bias types of the AR model should be considered as auxiliary inputs for this method. Therefore, in order to work with the CRNN, they are represented with 0s and 1s by the One Hot Encoding method.

- Exponential Bias = [1, 0, 0]
- Mean Shift Bias = [0, 1, 0]
- Oscillation Bias = [0, 0, 1]

In order to conduct experiments with the CRNN method, the bias type, as described above, is given as a categorical input to the time series data network. The network is expected to learn and estimate the target truncation point values better by considering this input in addition to the time series.

The architecture of the CRNN network with which experiments were carried out were taken as the same as the architecture of the LSTM network. The only difference was that the conditional recurrent layer was used as the first layer in the CRNN network. The same memory and computing time problems reported for the LSTM network in the previous section were also experienced for the CRNN network. Therefore, time series data with different lengths of 1000, 200 and 100 were also tried when working with the CRNN. Remaining configuration parameters of this network were also selected as the same as for the LSTM network.

CHAPTER 5

EXPERIMENTS AND RESULTS

In this chapter, firstly generation of datasets from the AR and M/M/1 simulation models, which will be used in training and testing the neural networks, is explained in Section 5.1, followed by the experimental settings for the two models in Section 5.2. The performance measures used in the study are defined in Section 5.3. Section 5.4 is allocated to hyperparameter tuning for the proposed machine learning methods. The experimental results for the three types of neural networks proposed for truncation point estimation are summarized in Section 5.5. Finally, the MLPR network, which is found to be successful in estimation, is compared with a conventional truncation point estimation method in Section 5.6.

5.1 Generation of Datasets

Two different models are used to generate datasets to represent the simulation output. The first is the AR model and the second is the M/M/1 model as described in Sections 3.2 and 3.3, respectively. This section will explain how we generated the datasets from these two models.

5.1.1 Autoregressive Model Datasets

In this work, zero-mean second order AR model datasets were used as suggested by White et al. (2000). The formulation of the zero-mean second-order AR model is as follows.

$$X_i = \phi_1 X_{i-1} + \phi_2 X_{i-2} + \varepsilon_i \tag{5.1}$$

where X_i is the AR model output, ϕ_1 and ϕ_2 are model coefficients and ε_i is the error, which is generated as a normally distributed random value with mean 0 and standard deviation 1. To represent the initial transient phase in steady-state simulations, three different types of bias are introduced to the AR model. These bias functions, which are also used by White et al. (2000), are given in Equations 5.2 through 5.4.

- Exponential Bias

$$B_i = Ce^{-0.005(i-1)} \tag{5.2}$$

- Mean Shift Bias

$$B_i = C/5 \tag{5.3}$$

- Oscillation Bias

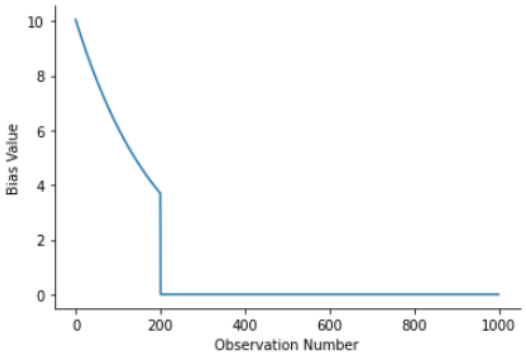
$$B_i = Ce^{-0.005(i-1)} \sin\left(\frac{i\pi}{200} + \frac{\pi}{2}\right) \tag{5.4}$$

where i is the observation number in the AR time series, B_i is the bias value for observation i and C is the bias coefficient. In order for the AR model to represent both the transient and the steady-state phases, B_i values of a certain type are added to the model as in Equation 5.5 during the transient phase. However, in the steady-state portion of the time series, B_i values are taken as zero so that there is no bias in the data. To illustrate the behavior of three bias functions, some sample time series are plotted in Figure 5.1 where the bias coefficient C is taken as 10 and the transient phase ends at observation number 200, which should be the truncation point.

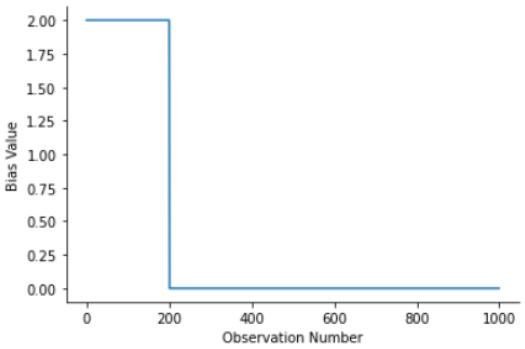
$$X_i = \phi_1 X_{i-1} + \phi_2 X_{i-2} + B_i + \varepsilon_i \tag{5.5}$$

where $B_i = 0$ for $i >$ truncation point.

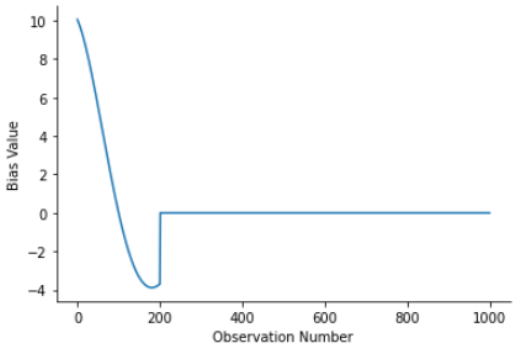
Samples used for training, cross validation, and testing of the neural network with the AR model are generated using Equation 5.5. In general, each sample of time series has a length of 1000 observations. As will be explained in Section 5.2, a total of 3000 samples are used for testing the AR model in 100 experiments each with 30 replications. Generated samples are used both as raw time series and after applying



(a) Exponential Bias Function



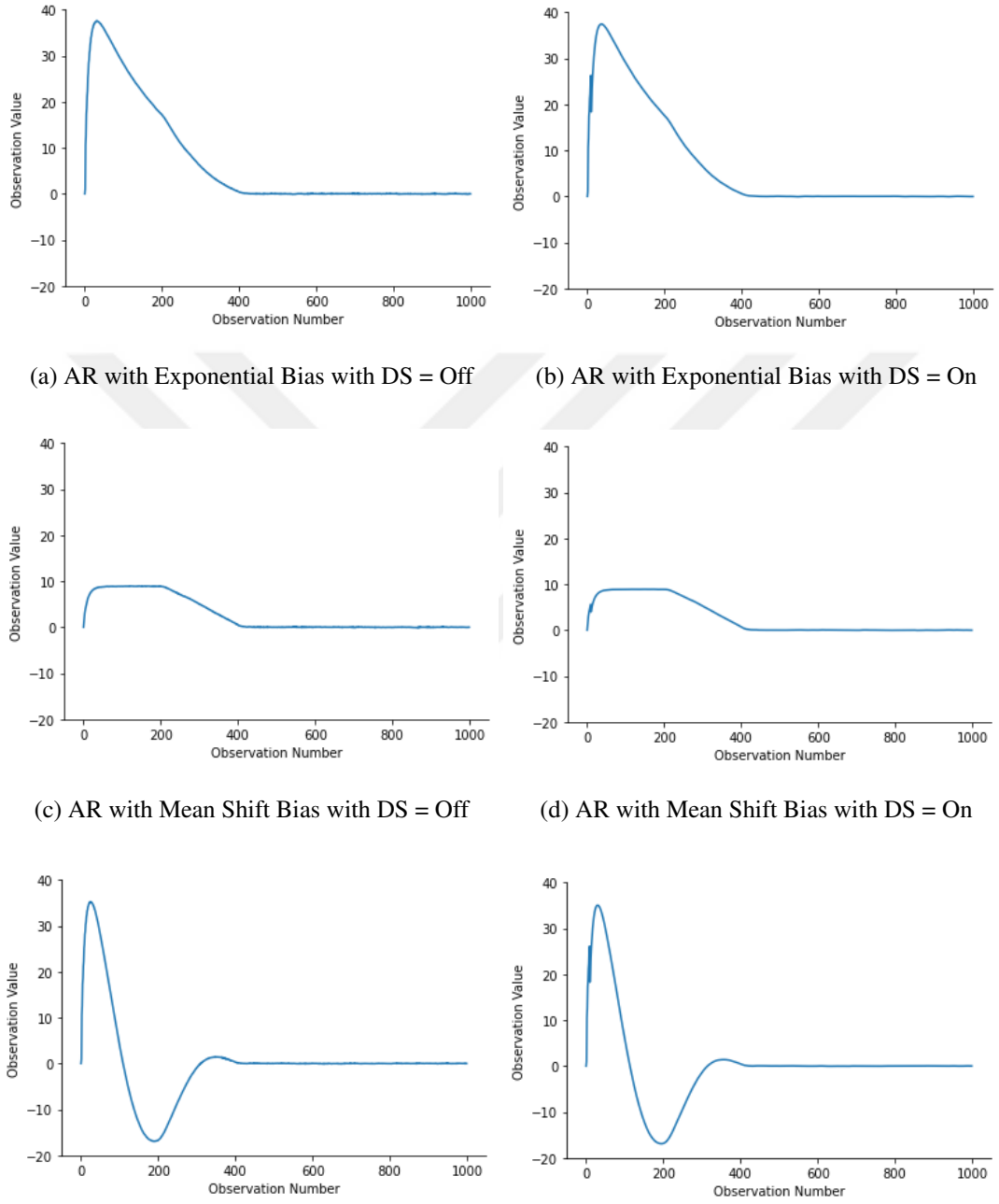
(b) Mean Shift Bias Function



(c) Oscillation Bias Function

Figure 5.1: Example Visuals for the Behavior of Three Bias Functions

data smoothing (DS) by taking moving averages of observations, as will be detailed in Section 5.2. In order to visualize the time series with three different types of bias, averages of 3000 test samples are plotted in Figure 5.2 for the raw time series (DS = Off) and moving averaged time series (DS = On).



(a) AR with Exponential Bias with DS = Off

(b) AR with Exponential Bias with DS = On

(c) AR with Mean Shift Bias with DS = Off

(d) AR with Mean Shift Bias with DS = On

(e) AR with Oscillation Bias with DS = Off

(f) AR with Oscillation Bias with DS = On

Figure 5.2: Generated Time Series Data for AR Model Testing with Random Model Parameters (Averages of 100 Experiments x 30 Replications)

5.1.2 M/M/1 Model Datasets

The M/M/1 model described in Section 3.3 is also used, as representative of queueing system models, for testing the proposed truncation point estimation methods and comparing them with a well known method in the literature. The main reason for selecting the M/M/1 model, on which many similar studies have been carried out in the field of simulation, is the ease of analysis of the results because the steady-state expected delay in queue, W_q, can be found as in Equation 5.6 given the arrival and service rates.

$$W_q = \frac{\lambda}{\mu\left(\mu - \lambda\right)} \tag{5.6}$$

where λ is the arrival rate and μ is the service rate.

In order to generate the delay in queue observations for M/M/1 datasets, instead of developing an actual simulation model, the well known recursive relation given in Equation 5.7 is used.

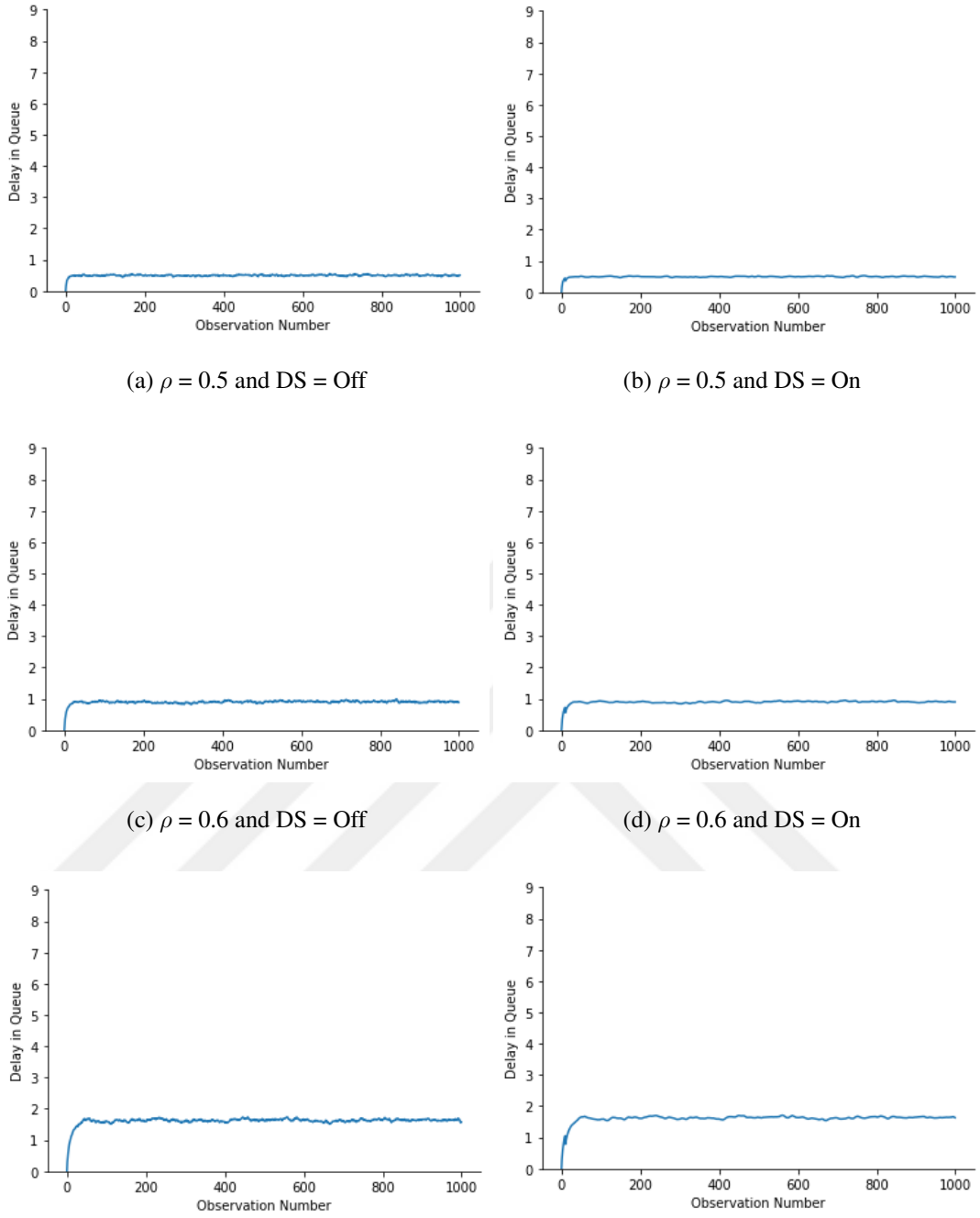
$$X_i = \max\left\{0, C_{i-1} - t_i\right\} \tag{5.7}$$

where

- A_i is the interarrival time between entities *i* and $i - 1$
- $t_i = t_{i-1} + A_i$ is the arrival time of entity *i* ($t_1 = 0$),
- S_i is the service time of entity *i*,
- X_i is the delay in queue of entity i ($X_1 = 0$), and
- $C_i = t_i + X_i + S_i$ is the service completion time for entity *i*

Unlike the AR model where the bias term is set to zero and this determines the truncation point, the truncation point in the M/M/1 model output is unknown and varies depending on the traffic intensity. Therefore, the M/M/1 datasets cannot be used for training and cross validation of the neural network; they are usable only for testing

purposes. As for the AR model, 3000 samples each of length 1000 observations are generated for the M/M/1 model for testing the neural network with different levels of traffic intensity. The M/M/1 model datasets used in this study can be visualized in Figure 5.3.



(a) ρ = 0.5 and DS = Off

(b) ρ = 0.5 and DS = On

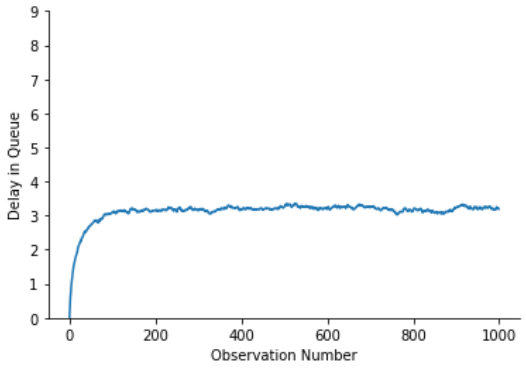
(c) ρ = 0.6 and DS = Off

(d) ρ = 0.6 and DS = On

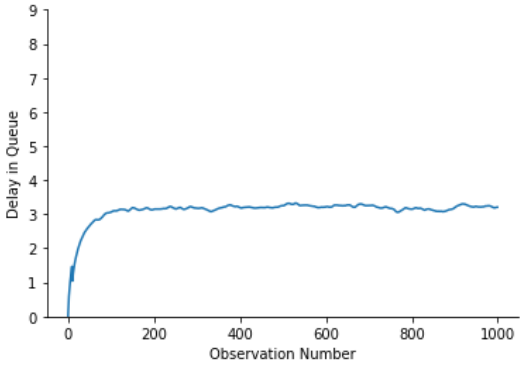
(e) ρ = 0.7 and DS = Off

(f) ρ = 0.7 and DS = On

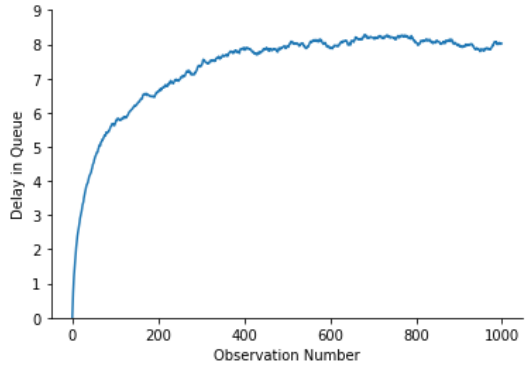
Figure 5.3: Generated Delay in Queue Data for M/M/1 Model Testing (Averages of 100 Experiments x 30 Replications)



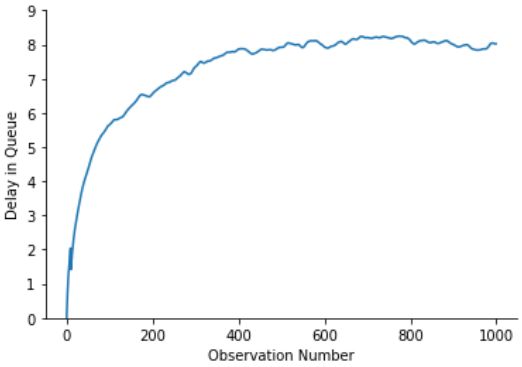
(g) $\rho = 0.8$ and DS = Off



(h) $\rho = 0.8$ and DS = On



(i) $\rho = 0.9$ and DS = Off



(j) $\rho = 0.9$ and DS = On

Figure 5.3: Continued

5.2 Experimental Settings

For both AR and M/M/1 models, there exist various parameters that affect the observation values in the generated datasets. In studying the performance of proposed solution methods, different parameter values are used for generating data from these two models. Firstly, the AR model parameter settings will be discussed within the framework of experimental design, followed by the M/M/1 model parameter settings.

5.2.1 Autoregressive Model Experimental Settings

The experimental settings for the AR model are summarized in Table 5.1, where model parameters are taken as factors in a designed experiment and selected values of parameters correspond to factor levels.

Table 5.1: Experimental Settings for the Autoregressive Model

Factor	**Number of Factor Levels**	**Factor Levels**
AP: Autoregressive Model Parameters	7	(0.6, 0.3) (0.9, 0.0) (-0.9, 0) (0.25, 0.5) (-0.25, 0.5) (0.75, -0.5) (-0.75, -0.5)
BT: Bias Type	3	Exponential Mean Shift Oscillation
BC: Bias Coefficient	3	5, 10, 15
TP5: Truncation Points	5	200, 250, 300, 350, 400
TP9: Truncation Points	9	200, 225, 250, 275, 300, 325, 350, 375, 400
DS: Data Smoothing	2	Off (individual observations) On (moving averages of 10 observations)

Seven different sets of model coefficients (AP) were used in generating the AR model data. While six of them were the values suggested by White et al. (2000), the first one was selected additionally to represent the positive autocorrelation pattern often encountered in queueing system outputs. Using many different sets of coefficients would be beneficial for the neural network to learn different data patterns and make better estimations in the training phase. In addition, training the neural network with a larger dataset containing all these different patterns would also help in the future when predicting data that the model never encountered before.

Simulation output encountered in real life have different types of initial bias (BT). For instance, the M/M/1 model exhibits an exponential bias before reaching to its steady-state. Therefore, introducing different bias types while training the neural network would improve its prediction performance. Three different types of bias functions were used in this study including exponential, mean shift, and oscillation bias functions. These bias types are described in Section 5.1.1 and Equations 5.2, 5.3, 5.4.

The bias coefficient (BC) is used as a multiplier to the bias value as seen in Equations 5.2, 5.3, 5.4. As can be seen from these equations, as the bias coefficient value increases, the bias value added to the data increases. Three different values were used as the bias coefficient (5, 10 and 15) to determine the magnitude of the bias values.

Truncation point (TP) determines up to which observation the bias values described in Section 5.1.1 will be added to the AR model data. Two different truncation point sets with five (TP5) and nine (TP9) discrete points were selected for the experiments. The range of the truncation point (200-400) was kept constant in order to make a better performance comparison, but TP9 provided a 'less discrete' set of truncation points compared to TP5. The purpose of keeping the truncation point range constant but increasing the number of points in TP9 is to explore the effect of the target variable values being closer to a continuous scale on the performance of the neural network.

Finally, instead of changing the value of a model parameter while generating the data, the moving average method was applied to the generated data in order to reduce the variance of the data. This created two different data types where data smoothing (DS) was off and on. DS = Off in Table 5.1 means that the raw data or individual observations were used as they are generated, while DS = On means that the moving averaged data were given as input to the neural network. The sample size for the moving averages was determined as 10 and they were calculated as in Equation 5.8.

$$\bar{X}_i = \frac{1}{10} \sum_{j=i}^{i+9} X_j \tag{5.8}$$

for $i = 1, 2, ..., 991$ where X_j for $i = 1, 2, ..., 1000$ is the j^{th} observation generated using the AR model and $\bar{X}_i$ is the respective moving average. For $i = 992, ..., 1000$, $\bar{X}_i$ is calculated only by using the available observations left.

As can be understood from Equation 5.8, length of the time series was set as 1000 observations for each replication. This means that, when the range of the truncation point (200-400) was considered, at least 600 steady-state observations were available in each sample given as input to the neural network. The replication length could have been much larger than 1000 for a simulation model, but this would significantly increase the memory and computing time required for training the neural network. Therefore, it was kept at a minimum level as long as the steady-state was reached long before the time series ended. Also, being able to determine the truncation point using a relatively short replication length was advantageous to spend less computing time for simulation replications as well.

The number of replications (R) to be used for training the neural network was selected such that the sample sizes for TP5 and TP9 were equilized. The number of combinations for the first three factor levels in the experimental design is 7 AP x 3 BT x 3 BC = 63. In case of TP9, when R was taken as 100 for each of the 63 x 9 TP9 = 567 factor combinations, a total training sample size of 56,700 was reached. To obtain the same sample size for TP5 with 63 x 5 TP5 = 315 combinations, the number of replications was determined as R = 56,700 / 315 = 180.

The two datasets for TP5 and TP9 each with a sample size of 56,700 were mainly used to evaluate the training performance of the neural network. For this purpose, five-fold cross validation was carried out. A dataset was divided into five equally sized folds by selecting samples randomly such that all factor combinations were equally represented in each fold having 11,340 samples. For each of the cross validation steps, one of these five folds was left out for validation and the remaining four folds were used for training the neural network.

In order to move one step forward and test the performance of the proposed machine learning methods, we generated a separate dataset which the neural network did not see before. For this purpose, for each BT, AP and BC values were chosen randomly from the discrete levels listed in Table 5.1 whereas the TP values were generated randomly between 200 and 400 in a continuous manner. The random selections were made with equal probability. In a single experiment, 30 replications were made each with a length of 1000 observations, and 100 different experiments were carried out

for each BT and data smoothing combination. In other words, 100 experiments x 30 replications = 3000 samples were generated for each BT for the AR model test dataset. For each test experiment, 30 replications were given as input to the neural network trained once by using all 56,700 samples.

5.2.2 M/M/1 Model Experimental Settings

The experimental settings for the M/M/1 model are summarized in Table 5.2 There are two factors in experiments using the M/M/1 data.

Table 5.2: Experimental Settings for the M/M/1 Model

Factor	**Number of Factor Levels**	**Factor Levels**
Traffic Intensity	5	0.5 0.6 0.7 0.8 0.9
DS: Data Smoothing	2	Off (individual observations) On (moving averages of 10 observations)

Five different values, 0.5, 0.6, 0.7, 0.8 and 0.9, were used for traffic intensity, defined as the mean arrival rate divided by the mean service rate. Queueing systems are known to reach steady-state in longer times as the traffic intensity increases, and it is needed to evaluate the performance of the proposed truncation point estimation methods under different intensity levels.

The moving averaging method for data smoothing mentioned in the experimental settings for the AR model data was also used in the M/M/1 experiments. Moving averages were calculated as in Equation 5.8.

In one test experiment of the M/M/1 model, 30 replications were made each with a length of 1000 observations, and 100 different experiments were carried out for each

traffic intensity and data smoothing combination. In other words, 100 experiments x 30 replications = 3000 samples were generated for each traffic intensity level for the M/M/1 model test dataset. For each test experiment, 30 replications were given as input to the neural network trained once by using all 56,700 samples of the AR model.

5.3 Performance Measures

We use the following measures in evaluating performance of the proposed truncation point estimation methods.

- Mean absolute percentage error (MAPE)
- Computing time for training
- Confidence interval for the steady-state expected value (CI-M) and bias
- Confidence interval for the coverage of CI-M (CI-C)
- Truncation point estimations

These performance measures are described and formulated below using the following notation.

- N : Number of replications (or samples given as input to the neural network in testing phase of the five-fold cross validation), $N = 11,340$.
- e : Experiment number, $e = 1, ..., k$ where $k = 100$ in testing by using newly generated data not used in training.
- j : Replication number, $j = 1, ..., n$ where $n = 30$ in a single test experiment.
- i : Observation number in a replication (or a sample given as input to the neural network in testing), $i = 1, ..., m$ where $m = 1000$.
- tp_j : The actual truncation point for replication *j*.
- TP_j : The estimated truncation point for replication *j*.

- X_{ije} : i^{th} observation generated in replication *j* of experiment *e* using the simulation model.
- μ : Steady-state expected value (of AR time series or delay in queue in M/M/1).

Mean absolute percentage error (MAPE)

MAPE is used to measure how well the machine learning methods can learn from the AR model data used for training. We use MAPE with the AR model data of 56,700 samples while cross validating the machine learning method using five folds where each fold contains $N = 11,340$ samples. The lower the MAPE value is, the closer the method's truncation point estimates are to the actual target values of the testing data during the five-fold cross validation. MAPE value for a single fold is calculated as

$$MAPE = \frac{100}{N} \sum_{j=1}^{N} \frac{|tp_j - TP_j|}{tp_j} \tag{5.9}$$

Computing time for training

In addition to the MAPE value, we also examine the training time of the machine learning method used.

Confidence interval for the steady-state expected value (CI-M) and bias

CI-M is used for both the AR and M/M/1 models in testing the neural network by using newly generated data not used in training. In order to understand whether or not a time series still has significant initialization bias left after the estimated truncation point, the observations before the estimated truncation point are deleted in each replication and a confidence interval for the steady-state expected value is constructed using the remaining observations. In addition, the bias between the expected value and the sample mean is also examined. CI-M and bias are computed for each experiment, using replications of that experiment. In constructing CI-M, the significance level is taken as $\alpha = 0.05$

Truncated average for replication *j* of experiment *e* is found as

$$\bar{X}_{je} = \frac{1}{m - TP_{je}} \sum_{i=TP_{je}+1}^{m} X_{ije} \tag{5.10}$$

Sample mean for experiment *e* based on truncated replication averages is

$$\bar{\bar{X}}_e = \frac{1}{n}\sum_{j=1}^{n} \bar{X}_{je} \tag{5.11}$$

Sample variance for experiment *e* is

$$s_e^2 = \frac{1}{n-1}\sum_{j=1}^{n} (\bar{X}_{je} - \bar{\bar{X}}_e) \tag{5.12}$$

$(1-\alpha)\%$ confidence interval for the steady-state expected value (of AR time series or delay in queue in M/M/1) for experiment *e* is constructed as

$$\bar{\bar{X}}_e \pm t_{n-1,\ 1-\alpha/2}\sqrt{\frac{s_e^2}{n}} \tag{5.13}$$

Bias for replication *j* of experiment *e* is

$$Bias_{je} = \left|\mu - \bar{X}_{je}\right| \tag{5.14}$$

Bias for experiment *e* is

$$Bias_e = \frac{1}{n}\sum_{j=1}^{n} Bias_{je} \tag{5.15}$$

Confidence interval for the coverage of CI-M (CI-C)

CI-C is also used for both the AR and M/M/1 models in testing the neural network by using newly generated data not used in training. Analyzing the coverage probability is crucial for evaluating the accuracy and reliability of statistical estimation methods. In this study, coverage is concerned with the number (or fraction) of experiments where CI-M covers the steady-state expected value (or population mean) of the time series. In addition to finding the coverage value, a CI-C is constructed again at a significance level of $\alpha = 0.05$ for the true probability that a CI-M covers its respective steady-state expected value.

$(1-\alpha)\%$ confidence interval for the coverage probability is constructed as

$$\hat{p} \pm z_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{k}} \tag{5.16}$$

where $\hat{p}$ is the fraction of *k* experiments in which the steady-state mean is covered in CI-M.

Truncation point estimations

Finally, we examine the estimated truncation points for the AR and M/M/1 model data newly generated for testing. In order to understand the spread of the estimations made by the method used for these two data types, we analyzed the average, minimum, and maximum values of the truncation point estimations.

In addition, the total number of replications in 100 experiments is counted where the estimated truncation point is greater than the run length. The truncated replication average should be based on a sufficiently large number of observations to satisfy the normality assumption of confidence interval construction. Assuming this number should be at least 30, a truncation point as large as 970 can be allowed in a time series of length 1000. Hence, if the estimated truncation point is larger than 970, we assume that this estimate exeeds the replication length. Such cases are excluded from our statistical calculations.

Moreover, distribution of the estimated truncation points are examined and compared for competing methods by means of histograms.

5.4 Hyperparameter Tuning for Proposed Machine Learning Methods

Machine learning methods involve various parameter settings. These parameter settings directly influence the performance of the neural network. In this case, the number of hidden layers, the number of nodes in hidden layers (hidden layer sizes), the activation function to be used in hidden layers, the solver for optimizing weight parameter values, and the maximum number of training iterations are the main parameters that need to be determined for setting up the neural network. These parameters must be manually set before training the network.

In order to maximize the network performance, all of those parameters must be configurated, which is called hyperparameter tuning. In this study, we performed hyperparameter tuning only for the MLPR network. The configuration determined here was also used for the other two networks. The first step of setting the parameter values is to determine the candidate values, which can be seen in Table 5.3

Table 5.3: Possible Hyperparameter Settings for MLPR

Parameter	**Number of Possible Parameter Values**	**Possible Parameter Values**
Solver function	2	Adam, Sgd
Activation function	2	ReLU, Identity
Hidden layer sizes	3	100, 200, 400
Maximum number of iterations	2	200, 300

Then, combinations of the possible parameter values are to be tried in training the network to see which one performs better. The 24 possible combinations of these parameters are to be evaluated using four different datasets, involving two different truncation point values (TP5 and TP9) and two different data smoothing options (DS = off and on). Hence, there exist 96 possible combinations for hyperparameter tuning experiments. Since this number is too large, we decided to start the experiment with one parameter in mind while including all combinations of the remaining parameters. Results of this first step are used to fix the value of the first parameter under consideration. Then, the next parameter is experimented with in a similar manner.

In order to select the MLPR network parameters, 56,700 AR model samples generated based on the experimental settings given in Section 5.2 are used. In these experiments, the truncation points are given as the points at which the bias is set to zero. The results presented in this section are five-fold cross validated for each of the combinations tried for possible configurations. The MAPE values and training computing times reported in this section are the averages over five folds.

Firstly, the maximum number of iterations is analyzed. In the experiments conducted with all parameter combinations, the average and maximum number of iterations during the training are examined. The actual number of iterations for training the MLPR network did not exceed 200 even for a single fold. Therefore, the maximum number of iterations is fixed at 200 since it does not create any restrictions, and the second value of 300 is discarded. However, the number of iterations for the final MLPR

network configuration will be checked again.

Next, the solver function selection is investigated. As seen in Table 5.3, we have two candidate functions for the solver. We experimented with Adam and Sgd functions including all remaining network configurations. Since no significant interactions are observed among the parameters, the average MAPE values for Adam and Sgd functions are tried to be found over all other configurations to select one of the two functions. The average MAPE values for Adam and Sgd functions calculated over 24 combinations (2 activation functions x 3 hidden layer sizes x 2 TP values x 2 DS options) are given in Table 5.4 together with the average computing times.

Table 5.4: Results for Solver Function Selection

Solver Function	**Average MAPE**	**Average Computing Time**
Adam	13.07	133.38
Sgd	NA	NA

When the average MAPE values in Table 5.5 are examined, we can conclude that only the Adam function gives valid results. The Sgd solver cannot converge and produce predictions even when the maximum number of iterations is set to 5000. Hence, the Adam function is chosen as the solver for the MLPR network.

As seen in Table 5.3, there are two candidates, Relu and Identity, for the activation function. Once the solver function is fixed, the MAPE values found are averaged over 12 parameter combinations (3 hidden layer sizes x 2 TP values x 2 DS options). The performance of the solver functions is summarized in Table 5.5.

Table 5.5: Results for Activation Function Selection (with Solver Function Adam)

Activation Function	**Average MAPE**	**Average Computing Time**
ReLU	9.01	177.24
Identity	17.13	70.02

According to Table 5.5, when ReLU activation function is used, the average MAPE

is significantly lower than that of Identity function. Although the average computing time with ReLU is about double of that with Identity, it is still reasonable. Therefore, considering the significant MAPE improvement, ReLU is selected as the activation function and Identity is discarded. For all results to be reported hereafter, the activation function is set as ReLU.

Lastly, the hidden layer architecture is studied. Since all remaining parameter settings are fixed, hidden layer selections are made using only the four datasets (2 TP values as TP5 and TP9 x 2 DS options as off and on). Considering the input layer size of 1000, which is defined by the length of the time series, three candidates are determined for the hidden layer size as 100, 200, and 400. The results obtained with a single hidden layer of these size candidates are given in Table 5.6.

Table 5.6: Results for Hidden Layer Size Selection with One Hidden Layer (with Solver Function Adam and Activation Function ReLU)

Hidden Layer Size	**Average MAPE**	**Average Computing Time**
100	9.33	88.90
200	8.95	133.09
400	8.76	309.72

According to Table 5.6, performances of three settings are similar. However, the hidden layer size of 100 is eliminated since this setting has the worst performance in terms of MAPE. Considering the tradeoff between MAPE and computing time, 200 nodes is selected as the size of the first hidden layer, and 400 nodes setting is also eliminated. Additional hidden layers with different sizes are considered to see whether or not increasing complexity of the MLPR network improves the performance.

Table 5.7: Results for Hidden Layer Size Selection with Two Hidden Layers (with Solver Function Adam and Activation Function ReLU)

Hidden Layer Sizes	**Average MAPE**	**Average Computing Time**
200, 50	6.82	131.33
200, 100	6.40	146.57
200, 150	6.35	162.66

When a second hidden layer is added to the MLPR network, performance of the network is improved as can be seen in Table 5.7. In order to enhance this performance improvement, we added a third hidden layer as well. After the trials given in Table 5.8, we determined the number of nodes in three hidden layers as 200, 150, and 100. We also tried two extreme configurations obtained by halving and doubling the number of nodels in all three layers to see if the network performance would change significantly. Even though hidden layer sizes of 400, 300, 200 result in a lower average MAPE, this architecture takes too much computing time for training the MLPR network.

Table 5.8: Results for Hidden Layer Size Selection with Three Hidden Layers (with Solver Function Adam and Activation Function ReLU)

Hidden Layer Sizes	**Average MAPE**	**Average Computing Time**
100, 75, 50	3.01	159.99
200, 150, 100	2.15	235.07
400, 300, 200	2.10	415.92

The performance of the network is improved with addition of the second and the third hidden layers. However, in order not to complicate the network anymore and because the MAPE values in Table 5.8 are satisfactory, the experiments are not continued by adding more hidden layers. Therefore, the MLPR network architecture is determined as three hidden layers with 200, 150, 100 nodes, considering the MAPE and computing time tradeoff.

As mentioned earlier, the maximum number of iterations for the selected configuration is checked, and all actual number of iterations are found to be below 200. The final MLPR network configuration is summarized below.

- Maximum number of iterations = 200
- Activation function = ReLU
- Solver function = Adam
- Hidden layer sizes = 200, 150, 100

In comparing performances of the MLPR network and the other two proposed networks, the same configuration was used for the LSTM and CRNN networks for the parameters common to all three networks. The experimental results given in the next section were obtained based on this configuration.

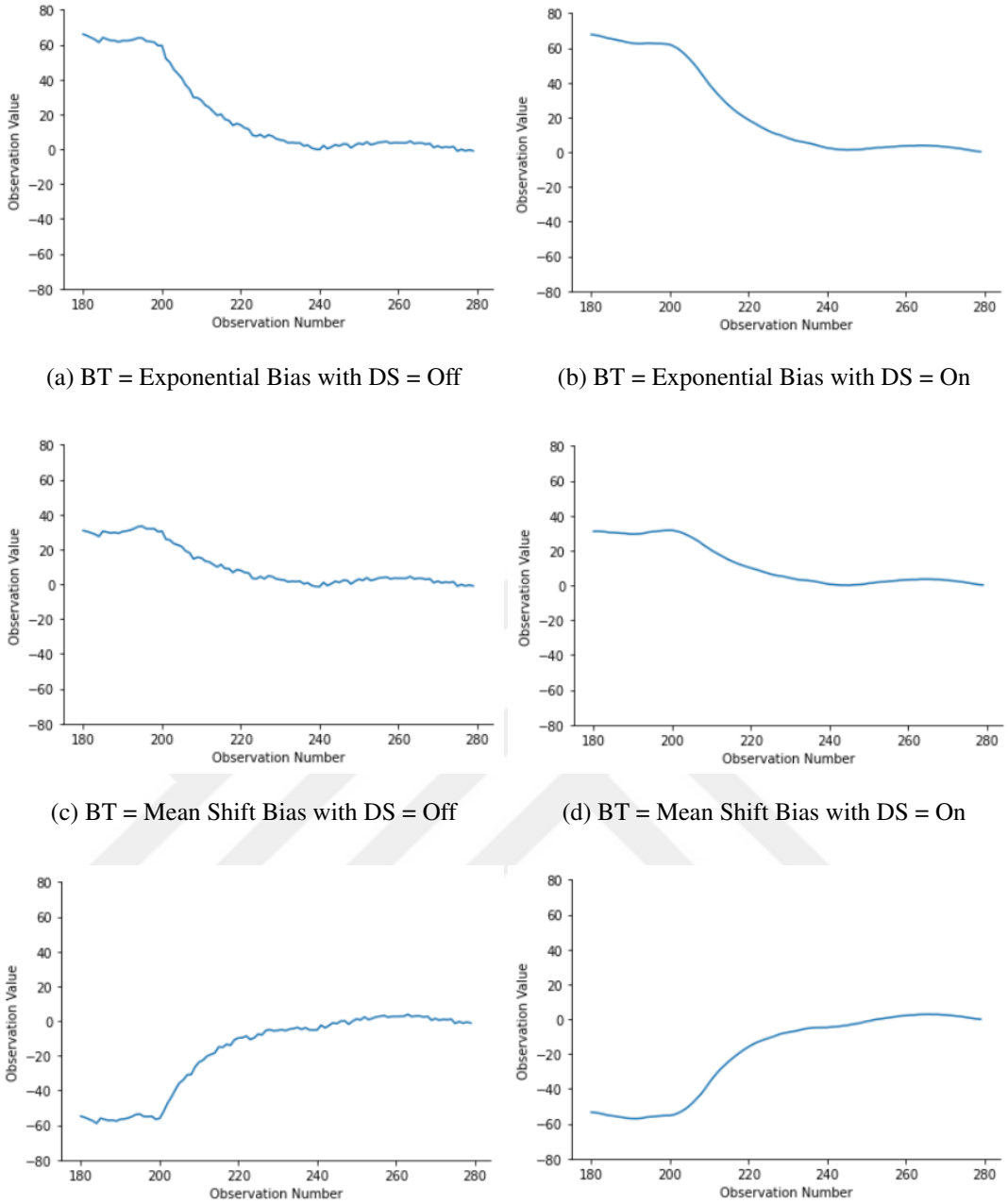
5.5 Computational Results for Solution Approaches

In this section, the performances of the three machine learning methods proposed are analyzed.

Based on the experimental settings described in Section 5.2 for the AR model, the dataset with 7 AP x 3 BT x 3 BC x 5 TP = 315 factor combinations each with 180 replications for TP5, and the dataset 7 AP x 3 BT x 3 BC x 9 TP = 567 factor combinations each with 100 replications for TP9 are used in the cross validation experiments, carried out separately for DS = off and on options. The number of replications are chosen considering the number of TP levels so that both datasets have the same size of 56,700 samples.

A dataset is divided into five equally sized subsets for five-fold cross-validation. For each subset, an equal number of samples (for example, 180/5 = 36 samples for TP5) are selected from each factor combination (for example, 7AP x 3BT x 3BC x 5TP = 315 combinations for TP5). Each one of these subsets is set apart for testing in a fold, and the remaining four subsets are used for training.

However, while generating the data in the AR model used, the effects of the added bias values do not end as sharply as in Figure 5.1, since each X_i is defined in terms of X_{i-1} and X_{i-2} values. In order to understand this better, plots of the time series were examined by focusing on the range in which the bias is present in the AR model data. As there are too many experimental combinations, here we present examples of these plots obtained by fixing some experimental factor levels. The highest bias coefficient cases (BC = 15) in the experimental settings are selected for this analysis. The motivation behind this choice is that, as the bias coefficient increases, the magnitude of the bias also increases, and the effect of the bias values added to the AR data can be seen more clearly. Also, the first AR model coefficient set (AP = 0.6, 0.3) is selected, and the last point where the bias values are not equal to zero is determined as 200. Plots of the AR time series for the range of 180^{th} through 280^{th} observations are given in Figure 5.4 for these settings. A second example with AP = 0.25, 0.5 is also given in 5.5.



(a) BT = Exponential Bias with DS = Off

(b) BT = Exponential Bias with DS = On

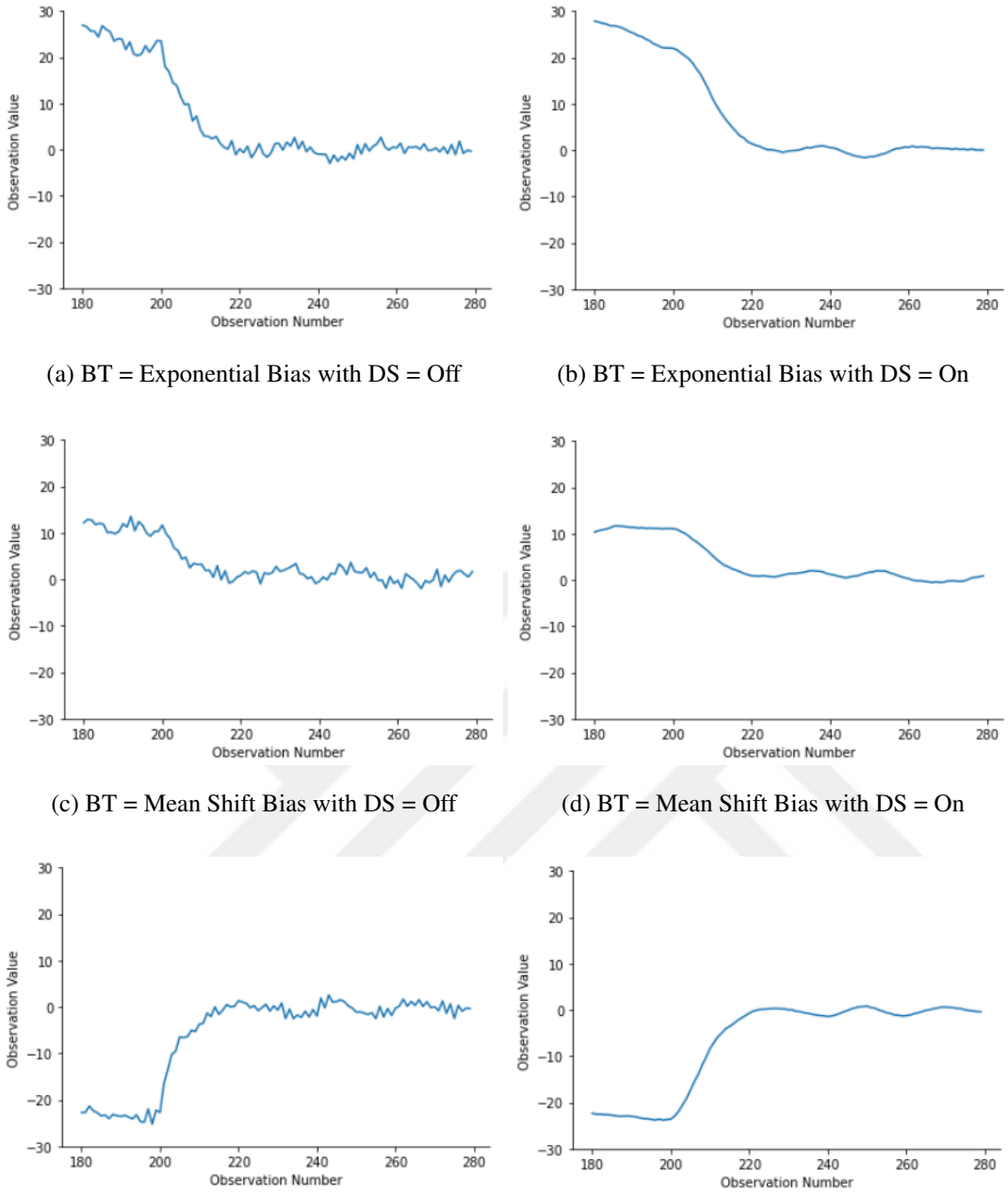
(c) BT = Mean Shift Bias with DS = Off

(d) BT = Mean Shift Bias with DS = On

(e) BT = Oscillation Bias with DS = Off

(f) BT = Oscillation Bias with DS = On

Figure 5.4: Visualization of the Bias Effects for AR Model Data (AP = 0.6, 0.3, BC = 15, TP = 200)



(a) BT = Exponential Bias with DS = Off

(b) BT = Exponential Bias with DS = On

(c) BT = Mean Shift Bias with DS = Off

(d) BT = Mean Shift Bias with DS = On

(e) BT = Oscillation Bias with DS = Off

(f) BT = Oscillation Bias with DS = On

Figure 5.5: Visualization of the Bias Effects for AR Model Data (AP = 0.25, 0.5, BC = 15, TP = 200)

As can be seen in Figures 5.4 and 5.5, although we reduce the bias values to zero after the 200^{th} observation, the effect of bias term in the model continues for a while. Since our purpose is to estimate the truncation point where steady-state is reached, we need

to determine the target value as the point at which the effect of the bias ends. Therefore, a modification must be made to the target variables given as input in training the neural network. After examining various plots similar to those given in Figures 5.4 and 5.5, it is observed that in general the AR time series reaches steady-state 50 observations later than the observation at which the bias is reduced to zero. Therefore, while training the neural networks in cross validation, target variable values for TP5 are given as 250, 300, 350, 400, and 450 instead of 200, 250, 300, 350, and 400, respectively. The same modification is also applied in training the networks using the TP9 dataset.

A summary of cross validation experiments conducted in this section is given in Table 5.9. In the "Estimation Option" column of the table, the machine learning method used is explained, including the neural network type and the content of the dataset. Note that the experiment is always repeated for DS = off and on options. "Truncation Point" values show how many different truncation point values are available in the dataset. The "Number of Models" column represents how many different neural networks are trained for each combination of estimation and truncation point options. For example, six different networks are trained for the combination of MLPR-S estimation option and TP5. For each bias type (exponential, mean shift, and oscillation), two different networks are trained with DS = off and on options, resulting in six different trained networks. While "Sample Size for Each Model" gives the dataset size according to our experimental settings, "Training Sample Size" and "Testing Sample Size" show the respective number of samples used in training and testing phases of five-fold cross validation.

The experimental results reported in the following subsections are produced by using the estimation options and sample sizes given in Table 5.9.

Table 5.9: TP Estimation Options and Sample Sizes for AR Model

Estimation Option	Truncation Point	Number of Models	Sample Size for Each Model	Training Sample Size	Testing Sample Size
MLPR-S	TP5	6	7AP x 3BC x 5TP x 180R = 18,900	15,120	3,780
Separately for each BT DS = off and DS = on	TP9	6	7AP x 3BC x 9TP x 100R = 18,900	15,120	3,780
MLPR-A	TP5	2	7AP x 3BT x 3BC x 5TP x 180R = 56,700	45,360	11,340
Together for all BT DS = off and DS = on	TP9	2	7AP x 3BT x 3BC x 9TP x 100R = 56,700	45,360	11,340
LSTM	TP5	2	7AP x 3BT x 3BC x 5TP x 180R = 56,700	45,360	11,340
Together for all BT DS = off and DS = on	TP9	2	7AP x 3BT x 3BC x 9TP x 100R = 56,700	45,360	11,340
CRNN	TP5	2	7AP x 3BT x 3BC x 5TP x 180R = 56,700	45,360	11,340
BT is also given as auxiliary input Together for all BT DS = off and DS = on	TP9	2	7AP x 3BT x 3BC x 9TP x 100R = 56,700	45,360	11,340

5.5.1 MLPR Experimental Results

The MLPR experiments described in this subsection are carried out on a computer with 32GB RAM, RTX 2080 Super 8GB as the GPU, and i7-8700K CPU @ 3.70GHz as CPU. Python Version 3.9 is used in implementing the MLPR network for training and testing purposes.

MLPR Cross Validation Test Results

With the predetermined MLPR configuration, we first conducted experiments to see whether or not the MLPR network could learn to estimate the truncation point from the AR model data. The motivation for training the neural network with the AR model data is that we can control up to which observation number in the time series the bias takes effect, when the bias term is set to zero, and then when the steady-state is reached, which gives us the TP estimate. Then, we can find the difference between this estimated TP value and the target TP value given to the network, and calculate the MAPE as a performance measure. Five-fold cross validation results of the MLPR-S and MPLR-A estimation options for the AR model, which are defined in Table 5.9, are given in Table 5.10 for TP5 and Table 5.11 for TP9. In order to analyze the MLPR network performance in more detail, we report the MAPE values separately for different target TP values as well as the overall MAPE. However, in terms of evaluating the performance, our main focus is the overall MAPE value.

When we examine the MAPE values for TP5 and TP9 in Tables 5.10 and 5.11, we make the following observations.

- Effect of data smoothing: For both TP5 and TP9, MAPE values for DS = on are significantly lower compared to those for DS = off. For the MLPR-S estimation options, the overall MAPE values range from about 4.5% to 6% when DS = on, whereas they remain consistently above 10% when DS = off. For TP9, they are over 20% for BT = mean shift or oscillation. MAPE values are almost halved for the MLPR-A estimation option as well. This is expected since data smoothing (taking moving averages of successive observations) reduces the variability in the time series and makes TP estimation easier.

Table 5.10: MAPE Results for TP5 in AR Model with MLPR (Averages for the Test Phase of Five-Fold Cross Validation)

		TP					
Estimation Option	**DS**	**200**	**250**	**300**	**350**	**400**	**Overall**
MLPR-S Exponential	off	12.0714	11.1019	10.7511	11.1083	12.0647	11.4195
MLPR-S Exponential	on	4.3787	4.5031	4.6099	4.6983	4.8578	4.612
MLPR-S Mean Shift	off	12.7126	10.0854	9.4512	9.8016	10.6528	10.5407
MLPR-S Mean Shift	on	4.8040	4.9612	4.8571	4.8927	5.1953	4.942
MLPR-S Oscillation	off	11.2772	10.0473	9.2931	9.7114	10.3046	10.1267
MLPR-S Oscillation	on	5.2357	5.8012	5.8647	5.8127	5.7108	5.6860
MLPR-A all BT	off	1.2945	1.3731	1.4342	1.2773	0.8275	1.2413
MLPR-A all BT	on	0.6507	0.7854	0.8591	0.8027	0.4918	0.7179

Table 5.11: MAPE Results for TP9 in AR Model with MLPR (Averages for the Test Phase of Five-Fold Cross Validation)

		TP									
Estimation Option	**DS**	**200**	**225**	**250**	**275**	**300**	**325**	**350**	**375**	**400**	**Overall**
MLPR-S Exponential	off	13.1605	12.1278	11.6331	11.7417	11.6318	11.6522	12.0283	12.2289	13.2052	12.1566
MLPR-S Exponential	on	4.5928	4.5975	4.4242	4.6511	4.6224	4.8697	4.8686	5.0407	5.4971	4.796
MLPR-S Mean Shift	off	40.6473	29.8823	22.0317	15.4462	12.7932	13.8748	15.8035	18.5372	22.3425	21.2621
MLPR-S Mean Shift	on	5.2838	4.7826	4.7575	4.5819	4.254	4.3626	4.4481	4.5897	5.0073	4.6742
MLPR-S Oscillation	off	37.5417	26.9317	19.3566	13.789	12.5454	13.0362	16.8976	19.7697	22.9569	20.3139
MLPR-S Oscillation	on	5.5996	5.5324	5.9997	6.2218	6.1969	6.8761	6.2176	6.2216	6.2417	6.123
MLPR-A all BT	off	3.0969	2.6648	2.6678	2.9282	2.5600	2.5414	2.1022	1.8551	1.9365	2.4837
MLPR-A all BT	on	1.1565	1.3743	1.4998	1.9228	1.6512	1.9148	1.1357	0.9880	0.6745	1.3686

- Effect of the estimation option: The MLPR-A estimation option shows a superior performance than all MLPR-S options. For TP5, the overall MAPE value stood at 1.24% when DS = off and is reduced to 0.72% when DS = on. The respective MAPE values for TP9 are 2.48% and 1.36%. These are much lower than MLPR-S MAPE values, which are over 4.5% even when DS = on. We can conclude that the MLPR-A network is more capable of deriving insights from the provided data and generating TP estimates that demonstrate a notable degree of proximity to target TP values. MLPR-S networks have to deal with a single bias pattern at a time, which should make TP estimation easier. However, the MLPR-S dataset size is one third of the entire dataset. We believe that the reason why the MLPR-A network shows much better performance is because it is trained using the entire dataset having three times as many samples as a MLPR-S dataset.

- Effect of the truncation point: When different TP target values are considered, the only pattern seen for the MLPR-S networks is that MAPE values are higher for the extreme TP targets (200 and 400) than those for the central TP values (closer to 300). On the other hand, the MLPR-A network results have the opposite pattern compared to MLPR-S. The MAPE values of the extreme TP values are lower than those of the central ones. The performance difference between MLPR-S and MLPR-A may again be due to the dataset size used in training the MLPR network.

- Effect of the bias type: For the case of TP5, there is not a significant difference in MAPE values among MLPR-S networks trained with exponential, mean shift, or oscillation bias types. For TP9, however, MLPR-S networks with exponential and oscillation bias types show mush poorer performance with MAPE values as high as 40% when the TP target is 200 and DS = off.

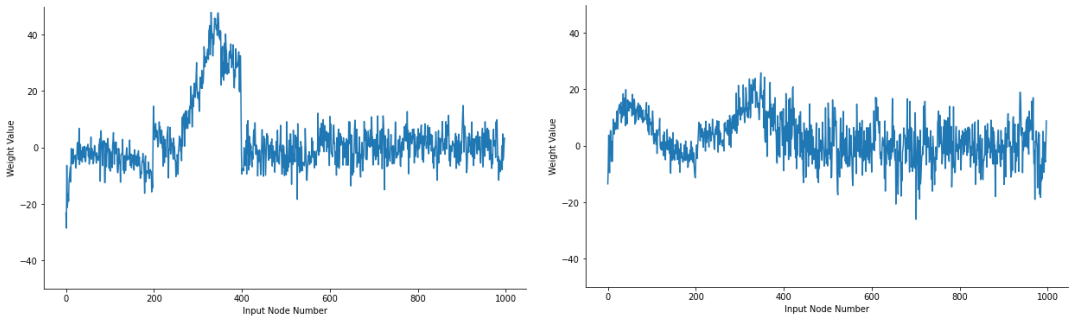
For the MLPR-S experiments, the computing time per fold of cross validation is in the range of 29-90 seconds, and the average computing time per fold is 46 seconds. For the MLPR-A experiments, the computing time per fold of cross validation is in the range of 142-398 seconds, and the average computing time per fold is 235 seconds.

Because significantly lower MAPE values were obtained from all four different net-

works of the MLPR-A estimation option, we decided to continue our work with this option. Furthermore, the outcomes were encouraging about the the network's ability to make predictions for previously unseen data. For examining the network weights and testing the selected estimation option based on previously unseen data, we came up with two versions of the MLPR-A option by re-training the network using all 56,700 samples. MLPR-TP5 version was obtained by training the network using the entire dataset of size 56,700 where generated samples had 5 discrete truncation points. Similarly, MLPR-TP9 version was obtained by training the network using the dataset of all samples involving 9 truncation points. Then, the network weights were analyzed in more detail and the networks were tested for truncation point estimation in the AR and M/M/1 models.

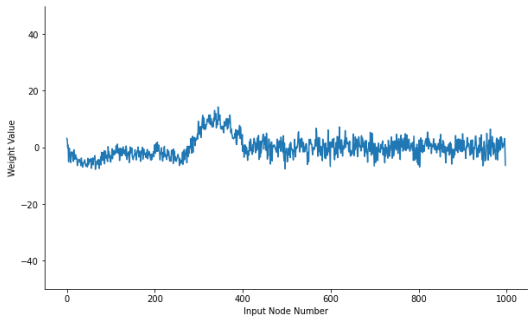
Examination of the MLPR Network Weights

Before moving on to the testing phase, we examined the weights of the MLPR network trained using 56,700 samples of data. From these weights, we can understand which input node the MLPR network considers the most likely when estimating the TP. There are 1000 nodes in the input layer, 200, 150, and 100 nodes in the three hidden layers, and 1 node in the output layer of the MLPR network. When we examined the weights, four different weight matrices were obtained of dimensions 1000x200, 200x150, 150x100, 100x1. Each weight matrix represents the transition from one layer to the next. By performing matrix multiplication on these matrices, we generated a weight matrix of dimensions 1000x1 (or a weight array of size 1000), which shows the effect of input data (time series) on the output (TP estimate). Such weight arrays of the MLPR-TP5 and MLPR-TP9 networks used in the testing phase are plotted in Figure 5.6.

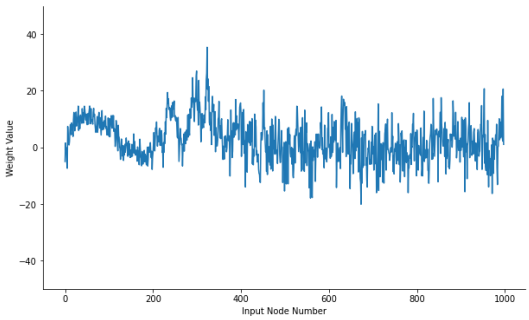


(a) MLPR-TP5 Weights, DS = Off

(b) MLPR-TP5 Weights, DS = On



(c) MLPR-TP9 Weights, DS = Off



(d) MLPR-TP9 Weights, DS = On

Figure 5.6: Product of Weight Matrices of the MLPR Network from Input Nodes to Output Node

As seen in Figure 5.6, the weights of the input values that are closer to the target TP values are larger and have more effect on TP estimation. The range of weights corresponding to the target TP values has an increasing pattern compared to weights corresponding to the remaining observation values, which is expected. This pattern is more pronounced in the case of TP5. Furthermore, the weights tend to hover around zero for the shared values that are prevalent in the steady-state condition across all the AR model data used in the MLPR network training.

MLPR Truncation Point Estimation Test Results for the AR Model

For testing the MPR-TP5 and MLPR-TP9 networks trained with the AR model data, a new dataset not used in training was generated as described in Section 5.2. For this purpose, for each BT, AP and BC values were chosen randomly from the discrete levels listed in Table 5.1 whereas the TP values were generated randomly between

200 and 400 in a continuous manner. The test dataset contains 30 replications (each of length 1000 observations) for a single experiment, and 3000 replications for 100 experiments for each BT. In order to better understand the testing process, the results obtained in a single experiment with MLPR-TP5 are given in Table 5.12.

For evaluating the results of a single experiment, performance measures used by White et al. (2000) were taken into account. The sample mean and standard deviation in Table 5.12 were found based on truncated averages of 30 replications using the TP estimate found for each replication. In addition, a confidence interval for the steady-state expected value (CI-M) was constructed, and the bias was calculated. A hypothesis test was also conducted on the equality of the sample mean and the expected value (or true mean). We fail to reject the H_0 that they are equal since p-values are above 0.05. This implies that the confidence interval constructed is likely to cover the true mean.

The results in Table 5.12 indicate that, for each bias type and DS option, the MLPR-TP5 network produces successful TP estimates. When the bias column is examined, the difference between the sample mean and the true mean is 0.0291 at the most. Hence, it can be said that TP estimates can eliminate the initialization bias in the time series. Also, the p-values of the hypothesis test are larger than the significance level of 0.05.

Although the MLPR-TP5 network performed well in a single experiment, more comprehensive analyses should be carried out for both MLPR-TP5 and MLPR-TP9 networks based on multiple experiments. Hence, 100 experiments were performed for the AR model each with 30 replications. The test results of these 100 experiments are summarized in Tables 5.13 and 5.14 for the two networks. After the statistics for each experiment are computed using truncated averages of 30 replications, averages are found over 100 experiments under the "AR Time Series" heading. Under the "Coverage" heading, the number of CI-M confidence intervals that cover the true mean is counted out of the 100 experiments. Then, a confidence interval for the coverage probability (CI-C) is constructed. Finally, the TP estimate statistics are given under the "TP Estimation" heading.

No weakness is observed for either network when the test results for the AR model

Table 5.12: TP5 Estimation Results for AR Model with MLPR-TP5 (Single Experiment, 30 Replications)

Bias Type True Mean (μ)	**DS**	**Sample Mean**	**Standard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**P-Value of One Sample t-Test**	**Avg. TP**	**Min. TP**	**Max. TP**
Exponential	Off	-0.007	0.2193	-0.0903	0.0763	0.0070	0.8643	380	299	452
$\mu = 0$	On	-0.0049	0.2213	-0.0889	0.0792	0.0049	0.906	388	310	462
Mean Shift	Off	0.0187	0.1706	-0.0461	0.0835	0.0187	0.5567	387	295	452
$\mu = 0$	On	0.0162	0.1764	-0.0508	0.0832	0.0162	0.6226	392	308	463
Oscillation	Off	-0.0225	0.1231	-0.0693	0.0243	0.0225	0.3291	383	298	451
$\mu = 0$	On	-0.0291	0.1188	-0.0742	0.0160	0.0291	0.1919	394	307	461

Table 5.13: TP Estimation Results for AR Model with MLPR-TP5 (100 Experiments)

		AR Time Series[1] (Average of 100 Experiments)					Coverage[2] (100 Experiments)			TP Estimation (3000 Replications)			
Bias Type True Mean (μ)	**DS**	**Sample Mean**	**Stan-dard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
Exponential	Off	-0.0006	0.1374	-0.0528	0.0516	0.0006	96	0.9215	0.9984	354	231	472	0
$\mu = 0$	On	-0.0002	0.1385	-0.0528	0.0524	0.0002	97	0.9365	1.0000	360	257	476	0
Mean Shift	Off	-0.0002	0.1520	-0.0579	0.0575	0.0002	99	0.9704	1.0000	352	222	484	0
$\mu= 0$	On	-0.0005	0.1529	-0.0586	0.0575	0.0005	98	0.9525	1.0000	360	228	474	0
Oscillation	Off	0.0034	0.1460	-0.0521	0.0589	0.0034	96	0.9215	0.9984	348	239	459	0
$\mu= 0$	On	0.0055	0.1474	-0.0504	0.0615	0.0055	96	0.9215	0.9984	361	252	471	0

[1] For each experiment, 30 replications each of length 1000 observations are made. For each replication, the truncated average is computed using observations after the estimated TP. Statistics for each experiment are computed using truncated averages of 30 replications. Then, averages are found over 100 experiments.

[2] For 100 experiments, the number of confidence intervals covering the true mean is counted. Then, a confidence interval for the coverage probability is constructed.

Table 5.14: TP Estimation Results for AR Model with MLPR-TP9 (100 Experiments)

		AR Time Series[1] (Average of 100 Experiments)					Coverage[2] (100 Experiments)			TP Estimation (3000 Replications)			
Bias Type True Mean (μ)	**DS**	**Sample Mean**	**Stan-dard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
Exponential	Off	0.0022	0.1404	-0.0511	0.0555	00022	97	0.9366	1.0000	397	293	453	0
$\mu = 0$	On	0.0016	0.1407	-0.0519	0.0550	0.0016	97	0.9366	1.0000	400	296	463	0
Mean Shift	Off	0.0061	0.1569	-0.0535	0.0657	0.0061	98	0.9526	1.0000	396	267	459	0
$\mu = 0$	On	0.0036	0.1567	-0.0559	0.0631	0.0036	97	0.9366	1.0000	401	281	474	0
Oscillation	Off	0.0094	0.1480	-0.0468	0.0656	0.0094	95	0.9073	0.9927	391	293	462	0
$\mu = 0$	On	0.0076	0.1481	-0.0486	0.0639	0.0076	96	0.9216	0.9984	400	294	476	0

[1] For each experiment, 30 replications each of length 1000 observations are made. For each replication, the truncated average is computed using observations after the estimated TP. Statistics for each experiment are computed using truncated averages of 30 replications. Then, averages are found over 100 experiments.

[2] For 100 experiments, the number of confidence intervals covering the true mean is counted. Then, a confidence interval for the coverage probability is constructed.

data are examined. Both MLPR-TP5 and MLPR-TP9 networks show great success in truncation point estimation for the AR model data. Under the "Coverage" heading, the number of CIs that cover μ is always over 95%. Under the "TP Estimation" heading, the numbers of TPs that are greater than the run length are all zero, which shows that the replication length of 1000 is sufficient. We can see that the most significant difference between the two networks is in the distribution of TP estimates. While the average of the TP estimates found by the MLPR-TP5 network is approximately 355, the estimates of the MLPR-TP9 network correspond to the 400^{th} observation on the average. Although both networks were trained with TP targets from the same range, the MLPR-TP9 network finds TP estimates at later observations than the MLPR-TP5 network. However, this difference does not adversely affect the performances of the networks for the AR model data.

MLPR Truncation Point Estimation Test Results for the M/M/1 Model

The M/M/1 model is used to test the MLPR networks to see if they are capable of estimating the TP for a different simulation model. Test data for delay in queue in the M/M/1 model are generated for five different traffic intensity values. DS = off and on options are also used in M/M/1 test experiments. The results of a single experiment are given in Table 5.15 where the MLPR-TP5 network is used for TP estimation. The table format is the same as of Table 5.12 for the AR model. These results are promising since the true mean values fall between lower and upper limits of the confidence intervals for all traffic intensity levels, and all p-values have values greater than 0.05. In addition, TP estimates become larger as the traffic intensity increases, especially when DS = off. This is expected because it is known that, as the traffic intensity increases, the variability in the system increases as shown by the standard deviation values, and the steady-state is reached at later times.

For the M/M/1 model, again 100 experiments were conducted with both MLPR-TP5 and MLPR-TP9 networks. The results are summarized in Tables 5.16 and 5.17. There are similarities in the results of the two networks. Bias values are small and coverage values are large enough for both networks when the traffic intensity is relatively lower. The confidence interval constructed for the coverage probability covers 95%. However, starting with the case where the traffic intensity is 0.8 and DS = On, the

Table 5.15: TP Estimation Results for Delay in Queue in M/M/1 with MLPR-TP5 (Single Experiment, 30 Replications)

Traffic Intensity (ρ) True Mean (μ)	**DS**	**Sample Mean**	**Standard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**P-Value of One Sample t-Test**	**Avg. TP**	**Min. TP**	**Max. TP**
$\rho = 0.5$	Off	0.5134	0.1262	0.4655	0.5614	0.0134	0.569	339	199	400
$\mu = 0.5$	On	0.5162	0.1258	0.4684	0.564	0.0162	0.4911	303	201	397
$\rho = 0.6$	Off	0.8605	0.17	0.7959	0.925	0.03949	0.2152	352	199	403
$\mu = 0.9$	On	0.8732	0.158	0.8132	0.9332	0.026800	0.365	307	185	429
$\rho = 0.7$	Off	1.5951	0.477	1.4139	1.7762	0.0379	0.6701	349	198	458
$\mu = 1.633$	On	1.624	0.4763	1.4431	1.8049	0.00899	0.9191	288	148	390
$\rho = 0.8$	Off	3.3062	1.347	2.7946	3.8178	0.10619	0.6728	367	199	556
$\mu = 3.2$	On	3.61	1.9883	2.8548	4.3651	0.40999	0.2714	342	140	790
$\rho = 0.9$	Off	7.2028	5.4315	5.0132	9.3923	0.89719	0.378	404	199	708
$\mu = 8.1$	On	7.0638	6.4489	4.5173	9.6103	1.0362	0.3908	368	123	896

coverage performance decreases for both networks. For the traffic intensity level of 0.9, the number of CI-M confidence intervals that cover μ decreases to 69 and 75 for MLPR-TP5 for DS = off and on. The same values are 64 and 25 for MLPR-TP9.

In parallel to coverage values decreasing with high traffic intensity, the number of TPs that are greater than the run length increases, which shows that the replication length of 1000 may be insufficient in these cases. This number should be zero or at least close to zero. When the traffic intensity is 0.9, MLPR-TP5 cannot find a TP estimate in 120 replications for DS = off and 89 replications for DS = on. However, these values go up to 667 and 1180 for MLPR-TP9. MLPR-TP9 starts experiencing this issue at traffic intensity level of 0.8. For MLPR-TP5, in the worst case, unpredicted replications only account for 4% of all 3000 replications. However, this value is about 39% for MLPR-TP9, which is unacceptable.

Although both networks show similar patterns of performance change, the numerical results indicate that the MLPR-TP5 network in general outperforms the MLPR-TP9 network. In addition to the coverage values and unpredicted replications discussed above, lower average bias values at high traffic intensity also indicate that the MLPR-TP5 network is more successful than the MLPR-TP9 network in TP estimation.

Since the performance of the MLPR networks was not at the desired level at high traffic intensity levels, we continued our work in this area. As the presence of unpredicted replications might be an indication of insufficient replication length, we decided to increase it from 1000 to 5000 observations. However, since both MLPR-TP5 and MLPR-TP9 networks were trained with 1000 observations, the length of the test data should also be 1000. Therefore, we reduced the simulation output of 5000 observations to the test data of length 1000 by batching the M/M/1 model observations. The batch size was taken as 5 and 1000 batch means were obtained to be used as the test data. Batching observations also resulted in reducing the variability in the simulation output.

The results obtained with batched test data for the traffic intensity levels of 0.8 and 0.9 are given in Tables 5.18 and 5.19.

The positive effect of increasing the simulation replication length and batching is

Table 5.16: TP Estimation Results for Delay in Queue in M/M/1 Model with MLPR-TP5 (100 Experiments)

		Delay in Queue[1] (Average of 100 Experiments)					**Coverage[2] (100 Experiments)**			**TP Estimation (3000 Replications)**			
Traffic Intensity (ρ) True Mean (μ)	**DS**	**Sample Mean**	**Standard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
$\rho = 0.5$	Off	0.4944	0.1022	0.4555	0.5332	0.0056	93	0.8799	0.9800	348	191	511	0
$\mu = 0.5$	On	0.4963	0.0982	0.4590	0.5336	0.0037	95	0.9072	0.9927	304	48	503	0
$\rho = 0.6$	Off	0.8877	0.2052	0.8097	0.9656	0.0122	92	0.8668	0.9731	347	187	598	0
$\mu = 0.9$	On	0.8928	0.1968	0.8181	0.9675	0.0072	90	0.8412	0.9587	299	81	547	0
$\rho = 0.7$	Off	1.6142	0.4712	1.4353	1.7932	0.0188	89	0.8286	0.9513	345	195	766	0
$\mu = 1.633$	On	1.6175	0.4557	1.4444	1.7906	0.0155	87	0.8040	0.9359	304	34	680	0
$\rho = 0.8$	Off	3.1081	1.3030	2.6128	3.6034	0.0919	90	0.8412	0.9587	359	161	955	4
$\mu = 3.2$	On	3.0849	1.2299	2.6175	3.5524	0.1151	83	0.7563	0.9036	314	18	954	3
$\rho = 0.9$	Off	7.2383	4.5674	5.4609	9.0157	0.8616	69	0.5993	0.7806	429	170	968	120
$\mu = 8.1$	On	7.3730	4.6272	5.5852	9.1608	0.7269	75	0.6651	0.8348	369	38	969	89

[1] For each experiment, 30 replications each of length 1000 observations are made. For each replication, the truncated average is computed using observations after the estimated TP. Statistics for each experiment are computed using truncated averages of 30 replications. Then, averages are found over 100 experiments. [2] For 100 experiments, the number of confidence intervals covering the true mean is counted. Then, a confidence interval for coverage probability is constructed.

Table 5.17: TP Estimation Results for Delay in Queue in M/M/1 Model with MLPR-TP9 (100 Experiments)

		Delay in Queue[1] (Average of 100 Experiments)					**Coverage[2] (100 Experiments)**			**TP Estimation (3000 Replications)**			
Traffic Intensity (ρ) True Mean (μ)	**DS**	**Sample Mean**	**Standard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
$\rho = 0.5$	Off	0.4904	0.0973	0.4534	0.5273	0.0096	90	0.8412	0.9588	346	190	701	0
$\mu = 0.5$	On	0.4924	0.1009	0.4541	0.5307	0.0076	93	0.8799	0.9800	334	202	679	0
$\rho = 0.6$	Off	0.8885	0.2114	0.8082	0.9688	0.0115	90	0.8412	0.9587	356	196	860	0
$\mu = 0.9$	On	0.8930	0.2271	0.8068	0.9793	0.0070	89	0.8287	0.9513	362	198	914	0
$\rho = 0.7$	Off	1.6223	0.5297	1.4207	1.8238	0.0107	92	0.8668	0.9732	386	190	899	10
$\mu = 1.633$	On	1.5994	0.5270	1.3981	1.8007	0.0336	90	0.8412	0.9588	420	193	964	33
$\rho = 0.8$	Off	3.1227	1.4659	2.5558	3.6895	0.0773	88	0.8163	0.9437	450	188	964	89
$\mu = 3.2$	On	2.9356	1.3679	2.3866	3.4847	0.2644	71	0.6210	0.7989	497	177	970	276
$\rho = 0.9$	Off	6.7786	4.1654	4.9412	8.6160	1.3214	64	0.5459	0.7340	562	191	970	667
$\mu = 8.1$	On	5.6435	3.0313	4.0813	7.2057	2.4565	25	0.1651	0.3349	556	205	959	1180

[1] For each experiment, 30 replications each of length 1000 observations are made. For each replication, the truncated average is computed using observations after the estimated TP. Statistics for each experiment are computed using truncated averages of 30 replications. Then, averages are found over 100 experiments. [2] For 100 experiments, the number of confidence intervals covering the true mean is counted. Then, a confidence interval for coverage probability is constructed.

Table 5.18: TP Estimation Results for Delay in Queue in M/M/1 Model with MLPR-TP5 (Batched Output, 100 Experiments)

		Delayin Queue[1] (Average of 100 Experiments)					**Coverage[2] (100 Experiments)**			**TP Estimation (3000 Replications)**			
Bias Type True Mean (μ)	**DS**	**Sample Mean**	**Stan-dard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
$\rho = 0.8$	off	3.1664	0.6004	2.9384	3.3945	0.0336	92	0.8668	0.9732	370	189	888	0
$\mu = 3.2$	on	3.163	0.6203	2.9275	3.3986	0.0370	90	0.8412	0.9588	318	0	931	0
$\rho = 0.9$	off	7.822	2.9045	6.7101	8.9339	0.2780	81	0.7331	0.8869	402	121	964	39
$\mu = 8.1$	on	7.7807	2.8546	6.6838	8.8775	0.3193	82	0.7447	0.8953	379	0	968	60

Table 5.19: TP Estimation Results for Delay in Queue in M/M/1 Model with MLPR-TP9 (Batched Output, 100 Experiments)

		Delayin Queue[1] (Average of 100 Experiments)					**Coverage[2] (100 Experiments)**			**TP Estimation (3000 Replications)**			
Bias Type True Mean (μ)	**DS**	**Sample Mean**	**Stan-dard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
$\rho = 0.8$	off	3.178	0.6779	2.9201	3.4359	0.0220	89	0.8287	0.9513	437	219	959	6
$\mu = 3.2$	on	3.1913	0.7611	2.9004	3.4821	0.0087	92	0.8668	0.9732	464	175	970	30
$\rho = 0.9$	off	7.4372	3.0586	6.159	8.7155	0.6628	69	0.5994	0.7806	596	210	969	450
$\mu = 8.1$	on	7.0558	2.7737	5.7889	8.3226	1.0442	50	0.4020	0.5979	573	91	970	823

clearly seen in Tables 5.18 and 5.19. With the batched simulation output, the performances of both MLPR-TP5 and MLPR-TP9 networks are increased at high traffic intensity levels. An increase is observed in the number of CIs that cover μ. We can conclude that the MLPR networks can find TP estimates such that truncated sample means obtained are closer to true means. The number of TPs greater than the run length also show a significant decrease when the batched output is used. Although the MLPR-TP9 network still has poor performance when the traffic intensity is 0.9, the MLPR-TP5 network seems to eliminate the deficiency in TP estimation with the help of increased replication length and batched M/M/1 data.

To summarize, the performances of both MLPR networks in experiments on the AR model data are promising. When we compare the two networks in the AR model experiments, there is not much difference between them. However, performances of the two networks differ when they are used on the M/M/1 model data. The MLPR-TP5 network is superior to the MLPR-TP9 network for the M/M/1 model. This may be because the TP5 version uses 'more discrete' TP target values than the TP9 version in training, and this may result in the MLPR-TP5 network learning better with more clear cut TP target values.

5.5.2 LSTM Experimental Results

The LSTM experiments described in this subsection are carried out on the same computer using the same Python version as the MLPR experiments.

LSTM Cross Validation Test Results

As with the MLPR network, the LSTM network was first cross validated using only the AR model data. The performance of the LSTM network was analyzed again by using five-fold cross validation.

In investigating whether or not the LSTM network can learn the given target variables with the AR model data, the LSTM network was first used with the same configuration as that of the MLPR network. However, as mentioned in Section 4.2.2, memory and computing time problems occurred since the LSTM network has a more complex structure than the MLPR network. For both DS = off and DS = on options, the LSTM network gave an out-of-memory error before completing 20 epochs. Therefore, in order to solve these problems, modifying the input data was unavoidable. The AR model time series was batched to generate new input data of lengths 200 and 100 as given in Equations 4.9 and 4.10.

The results obtained with the three different data lengths and respective hidden layer sizes with the configuration given in Section 4.2.2 are summarized in Table 5.20 for TP5 and Table 5.21 for TP9.

As can be seen in Tables 5.20 and 5.21, the overall MAPE values for LSTM exhibit unacceptably high levels. In both tables, when the MAPE values are examined separately for individual target TP values, we observe that the MAPE values are lower for the central target variable values. They are much higher for the extreme TP targets. Based on this observation, the TP estimations of the LSTM network were examined more closely, and it was discovered that the LSTM network found very similar TP estimates around the central target value for all samples in the dataset. Therefore, we concluded that the LSTM network could not learn the target variable values from the data, with our selected configurations.

The averages of the training times per fold for the LSTM network are approximately

Table 5.20: MAPE Results for TP5 in AR Model with LSTM (Averages for the Test Phase of Five-Fold Cross Validation)

			TP					
Input Length	**Hidden Layer Sizes**	**DS**	**200**	**250**	**300**	**350**	**400**	**Overall**
1000	200, 150, 100	off	NA	NA	NA	NA	NA	NA
1000	200, 150, 100	on	NA	NA	NA	NA	NA	NA
200	40, 30, 20	off	40.0144	16.6787	0.0103	12.4910	22.2142	18.2817
200	40, 30, 20	on	37.5998	14.6665	1.7144	14.0001	23.5557	18.3073
100	20, 15, 10	off	40.0572	16.7143	0.0409	12.4643	22.1904	18.2934
100	20, 15, 10	on	38.8368	15.6973	0.8309	13.2270	22.8684	18.2921

Table 5.21: MAPE Results for TP9 in AR Model with LSTM (Averages for the Test Phase of Five-Fold Cross Validation)

Input Length	Hidden Layer Sizes	DS	TP									Overall
			200	225	250	275	300	325	350	375	400	
1000	200, 150, 100	off	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
1000	200, 150, 100	on	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
200	40, 30, 20	off	40.0812	27.3465	16.7343	7.7548	0.0580	6.6125	12.4493	17.5993	22.1771	16.7570
200	40, 30, 20	on	37.9758	25.4325	14.9798	6.1352	1.4459	8.0161	13.7651	18.8378	23.3468	16.6595
100	20, 15, 10	off	40.1026	27.3660	16.7522	7.7712	0.0733	6.5983	12.4359	17.5867	22.1652	16.7613
100	20, 15, 10	on	40.1906	27.4460	16.8255	7.8389	0.1361	6.5396	12.3809	17.5349	22.1163	16.7788

2840 and 1460 seconds for the hidden layer size 40, 30, 20 and 20, 15, 10, respectively. Computing times for 200, 150, 100 are not given since 20 epochs are not completed. Compared to MLPR, the computing time performance also lags behind for the LSTM network.

To summarize, the LSTM network did not perform well with the specified configurations. The average MAPE values of five folds could not drop below 16% with different input data lengths. As these results were not at the desired level, test experiments were not conducted with the LSTM network on the complete sets of AR model data and M/M/1 model data. However, considering the possibility that the test performance could be better than the training performance, some limited test runs were made on the M/M/1 model data. When the LSTM network is tested on M/M/1 model data with traffic intensities of 0.5 and 0.9, LSTM estimated all the truncation points as the average of the target values as in cross validation tests of the AR model data. This supported our conclusion that the LSTM network could not learn how to estimate the truncation point.

5.5.3 CRNN Experimental Results

The CRNN experiments described in this subsection are carried out on the same computer using the same Python version as the MLPR experiments.

CRNN Cross Validation Test Results

The CRNN network is studied in a framework similar to the LSTM network. The only difference from the LSTM network is that the first layer of the hidden layers is the conditional recurrent layer, and the input data has been made suitable for this network. As mentioned before, this change in the input was made to analyze whether the performance of the neural network can be improved by giving the bias types in the AR model as an auxiliary input to the network.

However, the memory and computing time problems experienced in the LSTM network were also faced in the CRNN network. When a time series of length 1000 and its bias type were given as input to the CRNN network, 20 epochs could not be completed while training the network. The solution for the LSTM network was also

applied to the CRNN network. The data length was reduced from 1000 to 200 and 100, and the hidden layer sizes were also decreased accordingly.

The results of five fold cross validation with different input data lengths and different hidden layer sizes are given in Table 5.22 for TP5 and 5.23 for TP9.

As seen in Tables 5.22 and 5.23, the performance of the CRNN network is not encouraging. Results are similar to those obtained with the LSTM network. Average MAPE values calculated for five folds always stay above 16%. When the TP estimations made by CRNN on the AR model data are examined in detail, all of the estimates are almost the same around the central target value. It seems that, like LSTM, CRNN also estimates the TP as the average of the given TP target values for all samples in the dataset. The network could not effectively learn truncation point estimation for the AR model data under investigation. Consequently, no further tests were conducted with the CRNN network using the AR and M/M/1 data.

The averages of the training times per fold for the CRNN network are approximately 2890 and 1440 seconds for the hidden layer size 40, 30, 20 and 20, 15, 10, respectively. Computing times for 200, 150, 100 are not given since 20 epochs are not completed. Compared to MLPR, the computing time performance also lags behind for the CRNN network.

5.6 Comparison of MLPR Results with MSER and MSER-5

5.6.1 MSER and MSER-5 Definition and Results

Definition and Formulation of MSER and MSER-5

MSER and MSER-5 are two methods widely used in literature to estimate the truncation point. They are also included in many studies that compare the performance of existing truncation point heuristics or proposed new methods. White et al. (2000) compared MSER and MSER-5 with three different truncation point heuristics. Hoad et al. (2008) examined the truncation point heuristics with the purpose of automating them and concluded that MSER-5 is the best method to accomplish that. Mokashi et

Table 5.22: MAPE Results for TP5 in AR Model with CRNN (Averages for the Test Phase of Five-Fold Cross Validation)

			TP					
Input Length	**Hidden Layer Sizes**	**DS**	**200**	**250**	**300**	**350**	**400**	**Overall**
1000	200, 150, 100	off	NA	NA	NA	NA	NA	NA
1000	200, 150, 100	on	NA	NA	NA	NA	NA	NA
200	40, 30, 20	off	38.8604	15.7170	0.8140	13.2123	22.8553	18.2918
200	40, 30, 20	on	40.4616	17.0513	0.3297	12.2115	21.9658	18.4040
100	20, 15, 10	off	40.0195	16.6829	0.0139	12.4878	22.2114	18.2831
100	20, 15, 10	on	40.0828	16.7357	0.0591	12.4483	22.1762	18.3004

Table 5.23: MAPE Results for TP9 in AR Model with CRNN (Averages for the Test Phase of Five-Fold Cross Validation)

Input Length	Hidden Layer Sizes	DS	TP									Overall
			200	225	250	275	300	325	350	375	400	
1000	200, 150, 100	off	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
1000	200, 150, 100	on	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
200	40, 30, 20	off	40.0127	27.2843	16.6773	7.7021	0.0091	6.6582	12.492	17.6396	22.2162	16.7435
200	40, 30, 20	on	40.0034	27.2759	16.6695	7.6950	0.0025	6.6644	12.4979	17.6450	22.2203	16.7415
100	20, 15, 10	off	40.0100	27.2818	16.6750	7.7000	0.0072	6.6600	12.4937	17.6412	22.2167	16.7428
100	20, 15, 10	on	34.6005	22.3641	12.1671	3.5388	3.8568	10.2664	15.8747	20.8233	25.2220	16.5237

al. (2010) compared N-skart and MSER-5 methods. Oh and Park (2015) compared their proposed solution method (exponential variation rate) with MSER-5 considering their effectiveness, consistency, and confidence. Similarly, we compared our proposed solution methods withnd MSER-5.

White (1997) introduced the MSER method to the literature with the name Marginal Confidence Rule (MCR). Then, by modifying this method, Spratt (1998) conducted studies for the MSER-5 method. White et al. (2000) mentions that "The MSER and MSER-5 determine the truncation point as the point that best balances the tradeoff between improved accuracy (elimination of bias) and decreased precision (reduction in the sample size) for the reserved series." Here, the reserved series means the data remained after deleting the portion from the first observation to the truncation point estimate. With this approach MSER and MSER-5 aim at minimizing the width of the confidence interval constructed for the true steady-state mean. However, White et al. (2000) also state that "as the series of reserved observations is sequentially correlated, the marginal confidence interval is not a valid estimator of the truncated mean. The marginal confidence interval is a measure of the homogeneity of the truncated series reserved for analysis." However, we compare our methods with MSER based on independent replications, hence the confidence intervals constructed for the true steady-state mean are valid.

Given a series $X_i, i = 1, ..., m$, White et al. (2000) formulates the MSER method as in Equation 5.17.

$$d^* = \arg min \left[\frac{1}{(m-d)^2} \sum_{i=d+1}^{m} \left(X_i - \bar{X}_{m,d} \right)^2 \right] \tag{5.17}$$

where d^* is the optimal truncation point, *m* is the replication length, $\bar{X}_{m,d}$ is the truncated sample mean found using the truncation point candidate *d*. Calculation of $\bar{X}_{m,d}$ can be found in Equation 5.18. For the selection of d^*, we tested as *d* the first 800 observations (0.8 times the replication length of 1000) so that d^* will not be close to the last observations in every trial.

$$\bar{X}_{m,d} = \frac{1}{(m-d)} \sum_{i=d+1}^{m} X_i \tag{5.18}$$

While MSER deals with individual observations, MSER-5 works with non-overlapping batch means of size 5, calculated from the individual observations for determining the truncation point. The formulation of the batching process is given in Equation 5.19.

$$Z_j = \frac{1}{5}\sum_{i=1}^{5} X_{5(j-1)+i} \tag{5.19}$$

For the MSER-5 method, X_i in Equation 5.17 must be replaced with Z_j.

MSER and MSER-5 Truncation Point Estimation Results

In order to better understand the relative performance of the methods we proposed as a solution to the initial bias problem, we examined the performances of MSER and MSER-5 methods on the same test data and within the same experimental framework. In making a comparison, the performance measures described in Section 5.3 are also used for the MSER and MSER-5 methods. Firstly, TP estimation results of MSER and MSER-5 methods for the AR model data are given in Table 5.24 and Table 5.25, respectively.

As with the results of the MLPR network, the main performance measure we consider in Tables 5.24 and 5.25 is whether or not the confidence intervals constructed for the steady-state mean in 100 experiments cover the true mean. MSER and MSER-5 methods perform well in estimating the truncation point of data with DS = on option. Almost all of the confidence intervals constructed for the coverage probability cover 0.95 for both methods. Only the MSER-5 result with Exponential Bias and DS = on does not cover this value, but the upper limit of the coverage confidence interval is close enough. However, the results of the DS = off lines in Tables 5.24 and 5.25 are not at the desired level. When DS = off, both MSER and MSER-5 methods generally perform better for the oscillation bias compared to other bias types. The MSER-5 coverage values for this case rises to 92, which is satisfactory.

Another vital performance measure is whether or not the truncation point value estimated by the method is greater than the run length. However, this metric is not applicable for the MSER and MSER-5 methods since we restrict the candidate truncation points up to the 800^{th} observation in the time series.

Table 5.24: TP Estimation Results for AR Model with MSER (100 Experiments)

		AR Time Series[1] (Average of 100 Experiments)					Coverage[2] (100 Experiments)			TP Estimation (3000 Replications)			
Bias Type True Mean (μ)	**DS**	**Sample Mean**	**Standard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
Exponential	off	0.0777	0.1716	0.0125	0.1428	0.0777	60	0.5039	0.6960	265	0	656	-
$\mu = 0$	on	0.0061	0.1431	-0.0483	0.0604	0.0061	95	0.9072	0.9927	326	205	794	-
Mean Shift	off	0.1262	0.1674	0.0626	0.1898	0.1262	60	0.5039	0.6960	197	0	676	-
$\mu = 0$	on	0.0060	0.1573	-0.0537	0.0657	0.0060	99	0.9704	1.0000	326	198	796	-
Oscillation	off	-0.0227	0.1637	0.0849	0.0394	0.0227	67	0.5778	0.7621	240	0	638	-
$\mu = 0$	on	0.0073	0.1529	-0.0508	0.0654	0.0073	97	0.9365	1.0000	316	205	790	-

[1] For each experiment, 30 replications each of length 1000 observations are made. For each replication, the truncated average is computed using observations after the estimated TP. Statistics for each experiment are computed using truncated averages of 30 replications. Then, averages are found over 100 experiments.

[2] For 100 experiments, the number of confidence intervals covering the true mean is counted. Then, a confidence interval for coverage probability is constructed.

Table 5.25: TP Estimation Results for AR Model with MSER-5 (100 Experiments)

		AR Time Series[1] (Average of 100 Experiments)					**Coverage[2] (100 Experiments)**			**TP Estimation (3000 Replications)**			
Bias Type True Mean (μ)	**DS**	**Sample Mean**	**Standard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
Exponential	off	0.0195	0.1419	-0.0344	0.0733	0.0195	73	0.6429	0.8170	313	190	785	-
$\mu = 0$	on	0.0135	0.1456	-0.0418	0.0688	0.0135	88	0.8163	0.9436	328	200	790	-
Mean Shift	off	0.0266	0.1609	-0.0345	0.0877	0.0266	74	0.6540	0.8259	296	0	770	-
$\mu = 0$	on	0.0130	0.1579	-0.0469	0.0730	0.0130	98	0.9525	1.0000	327	195	795	-
Oscillation	off	0.0049	0.1509	-0.0524	0.0622	0.0049	92	0.8668	0.9731	297	75	780	-
$\mu = 0$	on	0.0056	0.1530	-0.0525	0.0637	0.0056	98	0.9525	1.0000	318	200	795	-

[1] For each experiment, 30 replications each of length 1000 observations are made. For each replication, the truncated average is computed using observations after the estimated TP. Statistics for each experiment are computed using truncated averages of 30 replications. Then, averages are found over 100 experiments.

[2] For 100 experiments, the number of confidence intervals covering the true mean is counted. Then, a confidence interval for coverage probability is constructed.

Performance of MSER and MSER-5 methods for the M/M/1 data are given in Tables 5.26 and 5.27, respectively. Truncation point estimations made by MSER and MSER-5 methods for the M/M/1 data do not yield encouraging results. When the values under the "Coverage" heading are examined, no coverage value is as high as desired. The maximum number of CIs that cover μ are are 51 and 52, far from the target of a 95% CI. We can understand the reason why the coverage values are so low from the metrics under the "Delay in Queue" heading. We can see that the bias values are large, which means that there is significant difference between the true mean and the sample mean obtained by discarding the data before the estimated truncation point. As a result, the confidence intervals do not cover the true mean delay in queue. To summarize, these two methods could not show a successful performance for the M/M/1 model data with low coverage values.

The increased replication length and batching improved the performance of the MLPR network in the M/M/1 model tests with higher traffic intensity levels. We investigated whether or not the same situation was valid for the MSER and MSER-5 methods. Data with a length of 5000 observations were produced from the M/M/1 model, and the data length was reduced to 1000 by batching. When these new data were given as input to MSER and MSER-5, results in Tables 5.28 and 5.29 were obtained. Although the replication length was five times as long as the original, this did not improve the coverage performances of MSER and MSER-5 when the traffic intensity was 0.9 and contributed only a little when the traffic intensity was 0.8.

Once the neural networks are trained, the TP estimation computing time for a test sample is negligible for both AR and M/M/1 data. With MSER, the computing time for an AR sample is in the range of 0.0952-0.3024 seconds, and the average computing time per sample is 0.1028 seconds. The same figures are 0.0045-0.0265 and 0.0048 seconds with MSER-5. For M/M/1 samples, MSER estimation times are in the range of 0.1026-0.5097 seconds, and the average computing time per sample is 0.1138 seconds. The same values are 0.0048-0.0268 and 0.0049 seconds with MSER-5. It should be noted that no extra effort was spent to optimize the codes for MSER and MSER-5 computations.

Table 5.26: TP Estimation Results for Delay in Queue in M/M/1 Model with MSER (100 Experiments)

		Delay in Queue[1] (Average of 100 Experiments)					**Coverage[2] (100 Experiments)**			**TP Estimation (3000 Replications)**			
Traffic Intensity ($) ***True Mean*** ($\mu$)	**DS**	**Sample Mean**	**Standard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
$rho = 0.5$	off	0.4672	0.0867	0.4342	0.5001	0.0328	46	0.3623	0.5576	68	0	798	-
$\mu = 0.5$	on	0.4456	0.0935	0.4101	0.4812	0.0544	16	0.0881	0.2318	165	0	799	-
$rho = 0.6$	off	0.8235	0.1717	0.7583	0.8887	0.0765	35	0.2565	0.4434	89	0	799	-
$\mu = 0.9$	on	0.7856	0.1842	0.7157	0.8556	0.1144	10	0.0412	0.1587	182	0	799	-
$rho = 0.7$	off	1.4405	0.3776	1.2971	1.5840	0.1925	26	0.1740	0.3459	114	0	799	-
$\mu = 1.633$	on	1.3812	0.3908	1.2328	1.5297	0.2518	12	0.0563	0.1836	189	0	799	-
$rho = 0.8$	off	2.6371	0.9184	2.2883	2.9859	0.5629	15	0.0800	0.2199	148	0	799	-
$\mu = 3.2$	on	2.5611	0.9504	2.2001	2.9220	0.6389	10	0.0412	0.1587	202	0	799	-
$rho = 0.9$	off	6.6476	5.2387	4.6580	8.6372	1.4524	52	0.4220	0.6179	200	0	799	-
$\mu = 8.1$	on	6.6129	5.3243	4.5908	8.6350	1.4870	50	0.4020	0.5979	226	0	799	-

[1] For each experiment, 30 replications each of length 1000 observations are made. For each replication, the truncated average is computed using observations after the estimated TP. Statistics for each experiment are computed using truncated averages of 30 replications. Then, averages are found over 100 experiments.

[2] For 100 experiments, the number of confidence intervals covering the true mean is counted. Then, a confidence interval for coverage probability is constructed.

Table 5.27: TP Estimation Results for Delay in Queue in M/M/1 Model with MSER-5 (100 Experiments)

		Delay in Queue[1] (Average of 100 Experiments)					**Coverage[2] (100 Experiments)**			**TP Estimation (3000 Replications)**			
Traffic Intensity (ρ) True Mean (μ)	**DS**	**Sample Mean**	**Standard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
$\rho = 0.5$		0.4579	0.0893	0.4240	0.4918	0.0421	34	0.2471	0.4328	134	0	795	-
$\mu = 0.5$	on	0.4489	0.0919	0.4140	0.4838	0.4140	19	0.1131	0.2668	173	0	795	-
$\rho = 0.6$		0.8069	0.1768	0.7397	0.8740	0.0931	24	0.1562	0.3237	147	0	795	-
$\mu = 0.9$	on	0.7888	0.1809	0.7201	0.8575	0.1112	12	0.0563	0.1836	193	0	795	-
$\rho = 0.7$		1.4166	0.3810	1.2719	1.5613	0.2163	20	0.1216	0.2783	157	0	795	-
$\mu = 1.633$	on	1.3857	0.3839	1.2399	1.5315	0.2473	16	0.0881	0.2318	197	0	795	-
$\rho = 0.8$		2.6090	0.9543	2.2465	2.9714	0.5610	12	0.0563	0.1836	181	0	795	-
$\mu = 3.2$	on	2.5717	0.9583	2.2077	2.9357	0.6283	11	0.0486	0.1713	208	0	795	-
$\rho = 0.9$		6.6471	5.2683	4.6462	8.6479	1.4529	49	0.3920	0.5879	214	0	795	-
$\mu = 8.1$	on	6.6362	5.3632	4.5993	8.6731	1.4638	51	0.4120	0.6079	233	0	795	-

[1] For each experiment, 30 replications each of length 1000 observations are made. For each replication, the truncated average is computed using observations after the estimated TP. Statistics for each experiment are computed using truncated averages of 30 replications. Then, averages are found over 100 experiments.

[2] For 100 experiments, the number of confidence intervals covering the true mean is counted. Then, a confidence interval for coverage probability is constructed.

Table 5.28: TP5 Estimation Results for Delay in Queue in M/M/1 Model with MSER (Batched Output, 100 Experiments)

		Delayin Queue[1] (Average of 100 Experiments)					**Coverage[2] (100 Experiments)**			**TP Estimation (3000 Replications)**			
Bias Type True Mean (μ)	**DS**	**Sample Mean**	**Standard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
$\rho = 0.8$	off	2.9992	0.4831	2.9814	3.0170	0.2008	41	0.3136	0.5064	82	0	799	-
$\mu = 3.2$	on	2.9742	0.4940	2.9558	2.9926	0.2258	34	0.2472	0.4328	99	0	799	-
$\rho = 0.9$	off	6.9662	2.1346	6.8857	7.0468	1.1338	26	0.1740	0.3459	144	0	799	-
$\mu = 8.1$	on	6.9279	2.1414	6.8470	7.0088	1.1721	23	0.1475	0.3125	154	0	799	-

Table 5.29: TP5 Estimation Results for Delay in Queue in Batched M/M/1 Model with MSER-5 (Batched Output, 100 Experiments)

		Delayin Queue[1] (Average of 100 Experiments)					**Coverage[2] (100 Experiments)**			**TP Estimation (3000 Replications)**			
Bias Type True Mean (μ)	**DS**	**Sample Mean**	**Stan-dard Deviation**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Bias**	**# of CIs that cover μ**	**95% CI Lower Limit**	**95% CI Upper Limit**	**Avg. TP**	**Min. TP**	**Max. TP**	**# of TPs > run length**
$\rho = 0.8$	off	2.9533	0.4996	2.9342	2.9724	0.2467	31	0.2194	0.4006	133	0	795	-
$\mu = 3.2$	on	2.9507	0.5015	2.9318	2.9696	0.2493	28	0.1920	0.3680	136	0	795	-
$\rho = 0.9$	off	6.8927	2.1273	6.8107	6.9748	1.2073	21	0.1302	0.2898	174	0	795	-
$\mu = 8.1$	on	6.8888	2.1364	6.8075	6.9701	1.2112	21	0.1302	0.2898	175	0	795	-

5.6.2 Comparison of Truncation Point Estimation Results for MLPR, MSER, and MSER-5

In this section, a comparison of the MLPR, MSER, and MSER-5 solutions reported in the previous sections will be made. Only a few of the performance measures are selected in making these comparisons. In the previous sections where the results of each solution approach were shared in detail, the number of CIs that cover μ, the number of TPs that are greater than run length, and occasionally the bias were discussed for each method, as they are found more important than the other metrics. Comparisons in this section will be made by taking the same performance measures into account.

Comparison of the test results for the AR model are summarized in Table 5.30 for the MLPR-TP5, MLPR-TP9, MSER, and MSER-5 methods.

The performances of all four solution methods, regardless of the bias type, were close to each other, and successful results were obtained when DS = on. The two MLPR networks outperform the MSER and MSER-5 methods in terms of the bias and the number of CIs covering μ in scenarios where DS = off. Both MLPR-TP5 and MLPR-TP9 networks achieve very satisfactory results in terms of the coverage regardless of the DS option. However, MSER and MSER-5 results when DS = off go down to 65-70% from approximately 95% achieved when DS = on. It seems that the MSER and MSER-5 methods benefit more from data smoothing, and MSER-5 is in general better than MSER. No problems were observed for any of the solution methods in terms of the number of TPs greater than run length.

Based on these results, we can see from Table 5.30 that MLPR networks outperform MSER and MSER-5 methods, although the latter methods show satisfactory performance as well when DS = on. As for the two MLPR networks, there is no evidence in their performances to decide one network is better than the other as far as TP estimation for the AR model is concerned. We can state that both MLPR-TP5 and MLPR-TP9 networks passed this test with the AR model successfully.

Results of the four solution methods for the M/M/1 model are given in Table 5.31 when DS = off and Table 5.32 when DS = on.

Table 5.30: Comparative Results for AR Model

Bias Type & DS	Solution Method	Bias	Number of CIs that cover μ	# of TPs >run length
Exponential DS = Off	MLPR-TP5	0.0006	96	0
	MLPR-TP9	0.0022	97	0
	MSER	0.0777	60	-
	MSER-5	0.0195	73	-
Exponential DS = On	MLPR-TP5	0.0002	97	0
	MLPR-TP9	0.0016	97	0
	MSER	0.0061	95	-
	MSER-5	0.0135	88	-
Mean Shift DS = Off	MLPR-TP5	0.0002	99	0
	MLPR-TP9	0.0061	98	0
	MSER	0.1262	60	-
	MSER-5	0.0266	74	-
Mean Shift DS = On	MLPR-TP5	0.0005	98	0
	MLPR-TP9	0.0036	97	0
	MSER	0.0060	99	-
	MSER-5	0.0130	98	-
Oscillation DS = Off	MLPR-TP5	0.0034	96	0
	MLPR-TP9	0.0094	95	0
	MSER	0.0227	67	-
	MSER-5	0.0049	92	-
Oscillation DS = On	MLPR-TP5	0.0055	96	0
	MLPR-TP9	0.0076	96	0
	MSER	0.0073	97	-
	MSER-5	0.0056	98	-

Table 5.31: Comparative Results for M/M/1 Model When DS = Off

Traffic Intensity	Solution Method	Bias	Number of CIs that cover μ	# of TPs >run length
0.5	MLPR-TP5	0.0056	93	0
	MLPR-TP9	0.0096	90	0
	MSER	0.0328	46	-
	MSER-5	0.0421	34	-
0.6	MLPR-TP5	0.0122	92	0
	MLPR-TP9	0.0115	90	0
	MSER	0.0765	35	-
	MSER-5	0.0931	24	-
0.7	MLPR-TP5	0.0188	89	0
	MLPR-TP9	0.0107	92	10
	MSER	0.1925	26	-
	MSER-5	0.2163	20	-
0.8	MLPR-TP5	0.0919	90	4
	MLPR-TP9	0.0773	88	89
	MSER	0.5629	15	-
	MSER-5	0.5610	12	-
0.8 Batched	MLPR-TP5	0.0336	92	0
	MLPR-TP9	0.0220	89	6
	MSER	0.2008	41	-
	MSER-5	0.2467	31	-
0.9	MLPR-TP5	0.8616	69	120
	MLPR-TP9	1.3214	64	667
	MSER	1.4524	52	-
	MSER-5	1.4529	49	-
0.9 Batched	MLPR-TP5	0.2780	81	39
	MLPR-TP9	0.6628	69	450
	MSER	1.1338	26	-
	MSER-5	1.2073	21	-

Table 5.32: Comparative Results for M/M/1 Model When DS = On

Traffic Intensity	Solution Technique	Bias	Number of CIs that cover μ	# of TPs run length
0.5	MLPR-TP5	0.0037	95	0
	MLPR-TP9	0.0076	93	0
	MSER	0.0544	16	-
	MSER-5	0.0414	19	-
0.6	MLPR-TP5	0.0072	90	0
	MLPR-TP9	0.0070	89	0
	MSER	0.1144	10	-
	MSER-5	0.1112	12	-
0.7	MLPR-TP5	0.0155	87	0
	MLPR-TP9	0.0336	90	33
	MSER	0.2518	12	-
	MSER-5	0.2473	16	-
0.8	MLPR-TP5	0.1151	83	3
	MLPR-TP9	0.2644	71	276
	MSER	0.6389	10	-
	MSER-5	0.6283	11	-
0.8 Batched	MLPR-TP5	0.0370	90	0
	MLPR-TP9	0.0087	92	30
	MSER	0.2258	34	-
	MSER-5	0.2493	28	-
0.9	MLPR-TP5	0.7269	75	89
	MLPR-TP9	2.4565	25	1180
	MSER	1.4870	50	-
	MSER-5	1.4638	51	-
0.9 Batched	MLPR-TP5	0.3193	69	60
	MLPR-TP9	1.0442	50	823
	MSER	1.1721	23	-
	MSER-5	1.2112	21	-

According to Tables 5.31 and 5.32, for the M/M/1 model, MLPR-TP5 network is the solution method that gives the most successful results among the four methods. For each traffic intensity level, the MLPR-TP5 network yielded the best performance in almost all cases, both in terms of the bias and the number of CIs that cover μ. It performed either the best or near the best in terms of all three metrics shown in the tables.

Interestingly enough, the coverage results of the MLPR-TP5 network is not worse when DS = off compared to the case when DS = on, in particular for the high traffic intensity levels. When the M/M/1 model data with increased replication length and batching is used for the traffic intensity level of 0.8, the number of CIs that cover μ is 92 with DS = off option and 90 with DS = on option. The same numbers for the traffic intensity level of 0.9 are 81 and 69. We can conclude that data smoothing is not absolutely necessary for the MLPR-TP5 network to succeed.

The MLPR-TP5 network is not comparable with MSER and MSER-5 in terms of the number of TP estimations greater than run length, as this metric is not applicable for MSER and MSER-5. However, we can ignore this metric in this comparison, as the MLPR-TP5 network gives significantly better results in other performance measures. Indeed, for the MLPR-TP5 network with DS = off, this value is at most 4% (120 out of 3000 replications), which is observed for the traffic intensity level of 0.9. Increasing the replication length and batching reduced this value to 1% (39 out of 3000 replications).

The MLPR-TP9 network is dominated by the MLPR-TP5 network in terms of the M/M/1 model test performance. However, it still achieves more successful results than the MSER and MSER-5 methods. In general, we can conclude that the MLPR networks give more successful results for a queueing system such as M/M/1.

In summary, both MLPR-TP5 and MLPR-TP9 networks performed better in TP estimation than MSER and MSER-5 methods, particularly in the M/M/1 model tests. MSER and MSER-5 could not outperform the MLPR networks for almost any test data. Although MLPR-TP5 and MLPR-TP9 networks could not outperform each other in the AR model tests, the MLPR-TP5 network dominated the MLPR-TP9 network in the M/M/1 model tests with or without data smoothing.

5.6.3 Comparison of Estimated Truncation Point Distributions for MLPR, MSER, and MSER-5

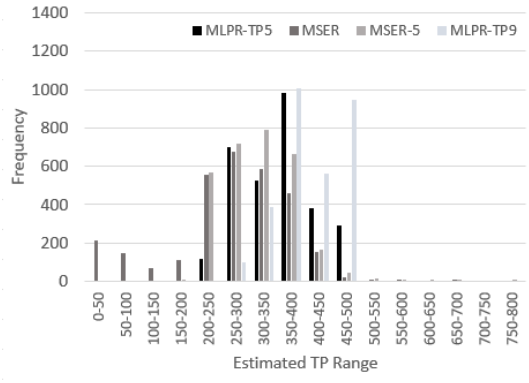
So far, we analyzed and compared the TP estimation performances of the three machine learning methods we proposed as well as the two conventional methods from the literature. This was done by using MAPE for cross validation and coverage of the confidence intervals constructed for the true steady-state means after the estimated TP values were used to discard the initialization bias. Although some statistics such as the average, minimum and maximum values were also reported for the TP estimates, we need to examine their distributions to better understand the performance differences of the competing methods.

To examine the distributions of the TP estimates from which the results in Tables 5.30, 5.31, and 5.32 are obtained, histograms of these estimates are plotted. Histograms of the TP estimates for the AR model are given in Figure 5.7 for the three bias types and two DS settings.

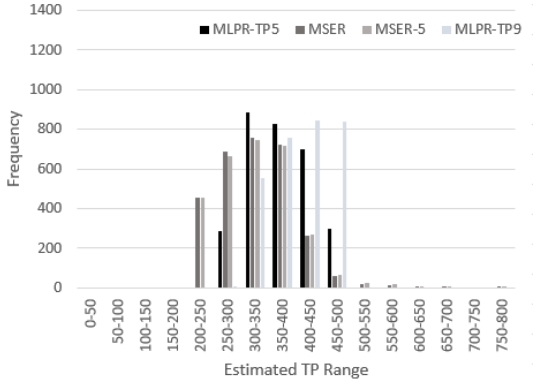
In the histograms for the AR model, the horizontal axis represents the bins of observation numbers in a replication. The vertical axis represents the frequency of TP estimates found by the solution methods. Since no TP estimate for the AR model exceeds 800 observations, the scale of the horizontal axis is narrowed to 0-800 range instead of the replication length of 1000 so that the histograms can be seen more clearly.

According to Figure 5.7, TP estimates found by all four methods when DS = on are distributed closer together (have a lower variability or spread) compared to the DS = off case where TP estimates have a higher variability or spread. Regardless of the bias type, all four solution techniques seem to find similar TP estimates with DS = on option with the exception of MLPR-TP9. TP estimates of the MLPR-TP9 network in general tend to be larger than those of the other methods. This is also seen in the average TP estimates of the MLPR-5 and MLPR-TP9 networks, which are 355 and 400, respectively.

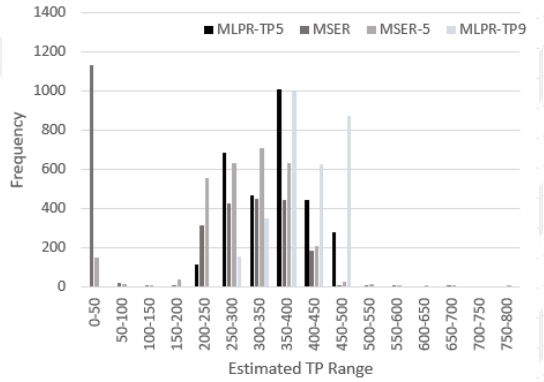
When DS = off, while MLPR-TP9 overestimates the TP values, MSER and MSER-5 methods come up with some very early TP estimates. This is the reason why the



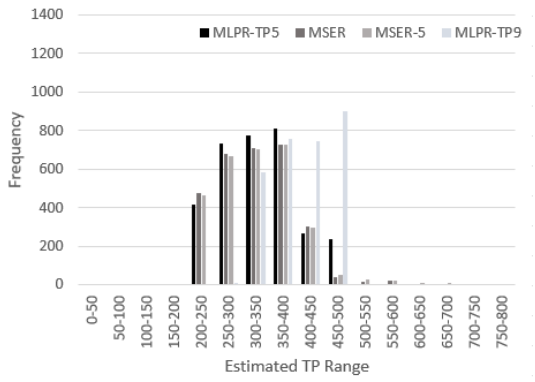
(a) BT = Exponential and DS = Off



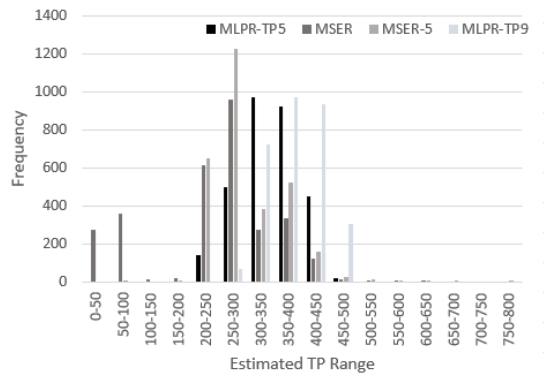
(b) BT = Exponential and DS = On



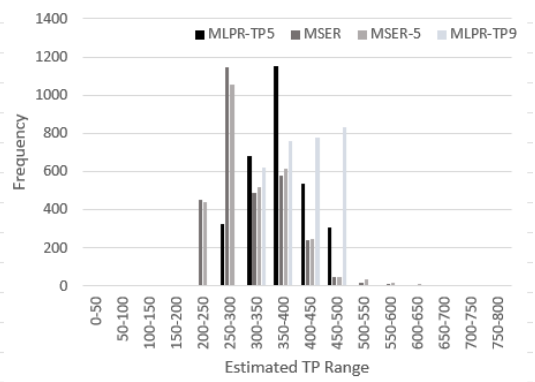
(c) BT = Mean Shift and DS = Off



(d) BT = Mean Shift and DS = On



(e) BT = Oscillation and DS = Off



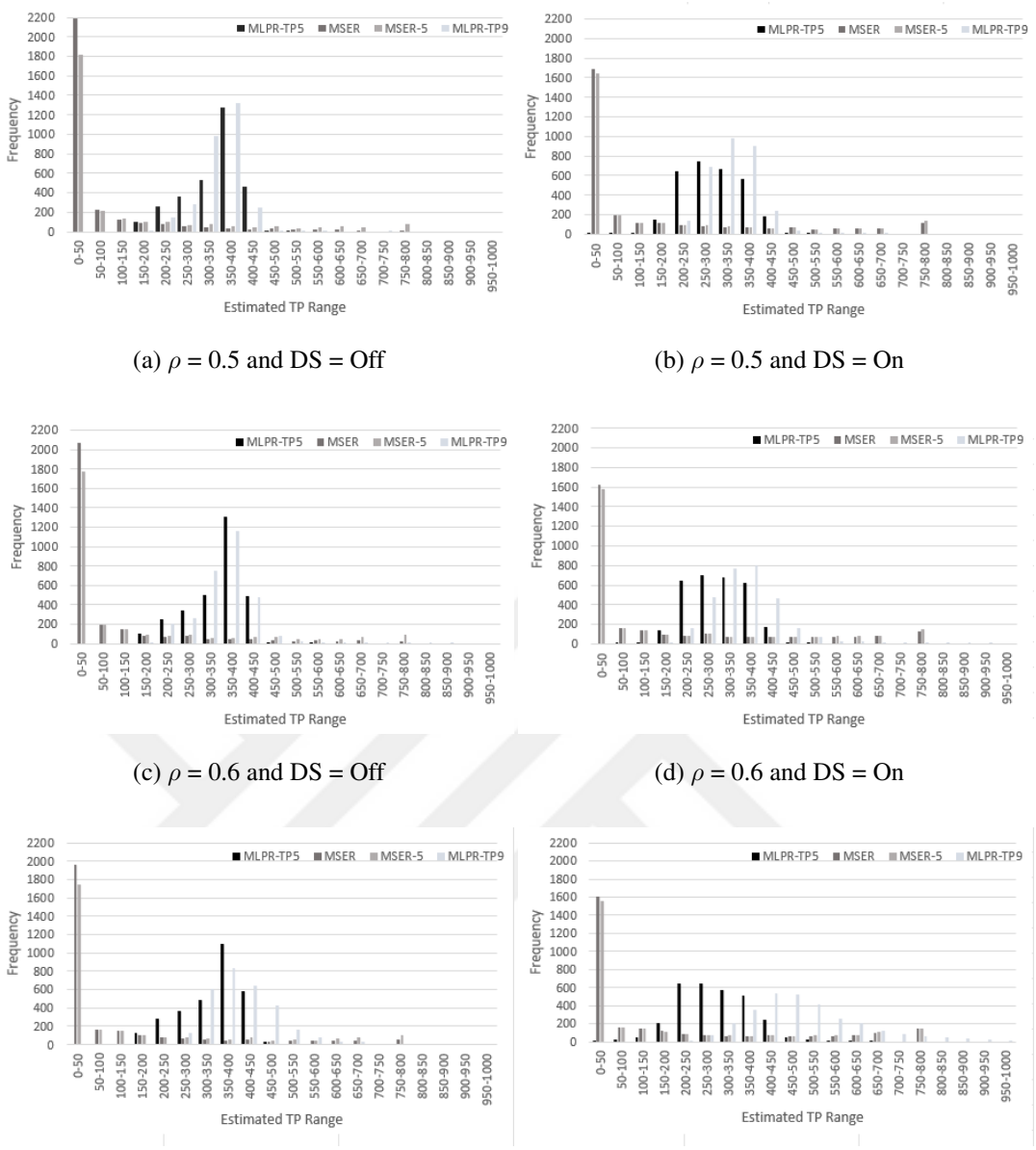
(f) BT = Oscillation and DS = On

Figure 5.7: Distribution of Estimated TP Values for AR Model (100 Experiments x 30 Replications)

coverage performance of these two methods is relatively lower in some scenarios where DS = off.

The MLPR-TP5 network, on the other hand, produce TP estimates in accordance with the TP range used in generating the AR model data, regardless of the DS setting. This explains the superior coverage performance of the MLPR-TP5 network.

Histograms of the TP estimates for the M/M/1 model are given in Figure 5.8 for the five traffic intensity levels and two DS options.



(a) $\rho = 0.5$ and DS = Off

(b) $\rho = 0.5$ and DS = On

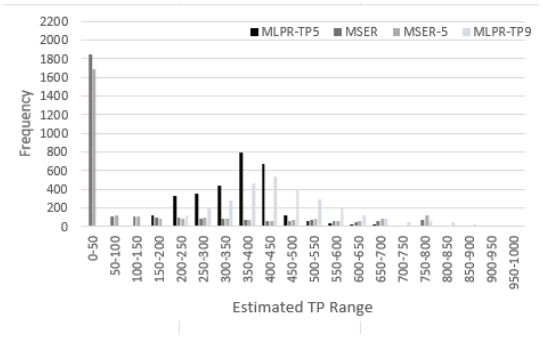
(c) $\rho = 0.6$ and DS = Off

(d) $\rho = 0.6$ and DS = On

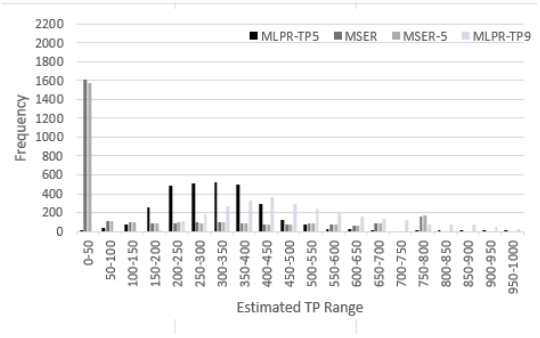
(e) $\rho = 0.7$ and DS = Off

(f) $\rho = 0.7$ and DS = On

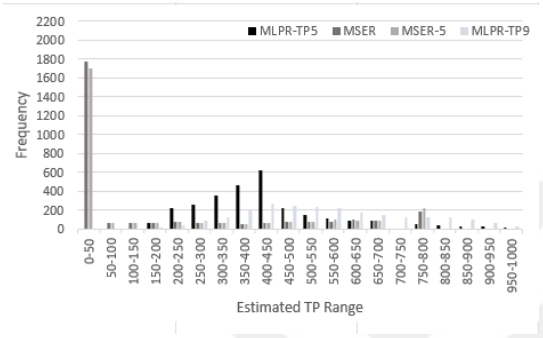
Figure 5.8: Distribution of Estimated TP Values for M/M/1 Model (100 Experiments x 30 Replications)



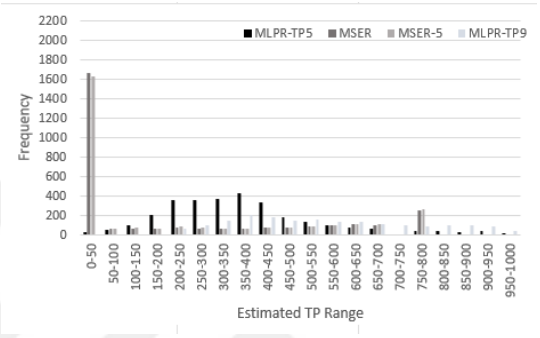
(g) ρ = 0.8 and DS = Off



(h) ρ = 0.8 and DS = On



(i) ρ = 0.9 and DS = Off



(j) ρ = 0.9 and DS = On

Figure 5.8: Continued

Compared to Figure 5.7 for the AR model, histograms in Figure 5.8 for the M/M/1 model exhibit a much higher spread. The most striking issue is that the TP estimates of the MSER and MSER-5 methods are mostly at the beginning of the data sequence. Underestimation of TP values is also observed in the AR model histograms, but it is more pronounced in the M/M/1 model. The poor performance of these two methods for the M/M/1 model can be attributed to this behavior. For all traffic intensity levels and both data smoothing options, the MSER and MSER-5 methods come up with premature TP estimates in the vast majority of cases.

The TP estimate distributions of MLPR-TP5 and MLPR-TP9 networks are in general similar for the M/M/1 model with low traffic intensity (especially 0.5, 0.6, and 0.7) and DS = off. These two networks seem to produce similar TP estimates. However, when the traffic intensity is 0.7 and higher and DS = on, the TP estimates of the MLPR-TP9 network have a more widespread distribution. When we compare

them with the estimates of the MLPR-TP5 network, we see that MLPR-TP9 estimates larger TP values closer to the end of the data sequence. This behavior of the MLPR-TP9 network is also supported by the larger number of TPs that are greater than run length compared to that of MLPR-TP5. The reason for this may be because the MLPR-TP9 network is trained with 'less discrete' target values, its TP estimates have higher spread, and it estimates that the steady-state is reached at a later point.

As the traffic intensity increases, the M/M/1 model is known to reach steady-state later. Both networks exhibit a similar behavior as the traffic intensity increases. Considering the maximum estimated TP values, the unpredicted TP values for low traffic intensity levels in the histograms began to be estimated for the traffic intensity level of 0.9. Similar results can be observed for the MSER and MSER-5 methods. For these two methods, the frequency of TP estimates in the range 750-800 increases for the traffic intensity level of 0.9.

The histograms of TP estimates support the inferences made from the numerical results given in the previous section. To summarize, when the histograms of the TP estimates are examined, no negative inferences are made regarding the estimates of the MLPR-TP5 network. However, problems are encountered with specific data types for the other methods. Therefore, we can conclude that the MLPR-TP5 network can be selected as the best solution approach in this study.

CHAPTER 6

CONCLUSIONS

Summary of the Work Done

This study addresses the initialization bias problem, which is encountered when analyzing outputs of steady-state simulations. Three different machine learning techniques, namely Multilayer Perceptron Regressor (MLPR), Long Short-Term Memory (LSTM), and Conditional Recurrent Neural Network (CRNN), were used to develop a solution approach for the initialization bias problem by estimating the truncation point in simulation output sequence.

In steady-state simulations, the output sequence obtained from the model is usually divided into two phases. The transient phase is the portion of the output data that contains the effects due to arbitrary initial conditions of the simulation. The transient phase is often followed by the steady-state phase where the system modeled reaches an equilibrium, and the output sequence and statistics stabilize. The output data collected during the transient phase are not representative of the system's steady-state, and cause bias in output statistics if included in the analysis. This is known as the initialization bias problem in steady-state simulations. In order to analyze the simulation output accurately, the transient phase must be eliminated from the data. To accomplish this, a truncation point must be determined, and the data before the truncation point should not be used in the output analysis. In this study, we employed three machine learning techniques for estimating the truncation point in steady-state simulation in an attempt to eliminate the initialization bias.

The autoregressive (AR) and M/M/1 queueing system models were used to generate the simulation output data. The original AR model generates a time series in steady-

state. For studying the transient phase and estimating the truncation point, some bias must also be included in the initial portion of the time series. Therefore, three different bias types were introduced to the AR model. Exponential, mean shift, and oscillation bias functions were added up to a certain point in the AR model time series. Through this approach, we could acquire the essential target truncation point values necessary for training the neural networks. The M/M/1 model data, on the other hand, do not have known truncation points but could be used in testing the neural networks.

The MLPR network is a special case of the multilayer perceptron, designed with adjustments to the output layer to enable regression tasks to predict or estimate continuous target variables. The LSTM is an advanced form of the recurrent neural networks (RNN) and employs specialized memory cells to capture short and long-term dependencies in sequences. The CRNN differs from conventional RNN networks by including a conditional layer. This unique layer enables the utilization of time-independent data as auxiliary input to the neural network. We could train these machine learning techniques with the AR model data as we knew the truncation point values. Our aim in this study was to analyze whether or not these neural networks could learn to estimate the truncation point and explore if we could eliminate the initialization bias problem with the help of these methods.

Performances of the three machine learning based solution approaches were examined by considering five different performance measures. The networks were five-fold cross validated using the AR model data. For the performance of each fold, mean absolute percentage error (MAPE) and computing time were taken into account. In the tests involving 100 experiments each with 30 replications, evaluations were made based on the confidence intervals for the steady-state expected values and bias, the coverage of these confidence intervals, and truncation point estimations. The confidence intervals were constructed after eliminating data before the estimated truncation point, to see if they cover the steady-state expected values.

Main Conclusions

Experiments were performed with similar configurations for all three neural networks. In the MLPR network cross validation, the MAPE values for truncation point estimates were found to be 0.7-2.5% depending on the experimental settings. How-

ever, similar performances could not be obtained for the LSTM and CRNN networks. The MAPE values for these two networks were approximately 17% and 18%. LSTM and CRNN estimated all truncation points as the average of the target truncation point values given to the network. Based on these results, we concluded that these two networks could not learn from the AR model data with the specified network configurations. Therefore, we continued to work only with the MLPR network in more detailed tests.

In order to analyze the performance of the MLPR network in more detail, it was compared with the conventional MSER and MSER-5 methods, which are well-known in the literature. MSER and MSER-5 aim to identify the truncation point as the optimal point considering the trade-off between enhanced accuracy (reducing the bias) and increased precision (reducing the variance) where the fixed sample size is most effectively balanced for the dataset at hand. Performances of the three methods were compared in terms of the previously mentioned performance measures.

While comparing these three methods, 100 experiments were conducted for each of the various experimental settings, using both AR and M/M/1 model test data. In the truncation point estimations made for the AR model, the performances of the MLPR, MSER and MSER-5 methods were similar in some scenarios, MLPR performed better in some other scenarios, but all three methods achieved successful results in general. The coverage of the confidence intervals for the steady-state expected values was in the ranges 96-99%, 60-99%, 73-98% for the MLPR, MSER and MSER-5 methods, respectively.

In the M/M/1 model tests, however, the MLPR network dominated the MSER and MSER-5 methods in terms of performance. At the low traffic intensity levels (0.5-0.7) in M/M/1, MSER and MSER-5 could not achieve the desired performance, with respective confidence interval coverages of 10-46% and 12-34%. The MLPR network (MLPR-TP5 version), on the other hand, produced coverages in the range 87-95%. Estimating the truncation point was more difficult for the MLPR network at the high traffic intensity levels (0.8-0.9) in M/M/1, hence the replication length was increased and batching was applied to the observations. In this case, MLPR achieved confidence interval coverages in the range 69-92%. With or without batching, the coverage

ranges for MSER and MSER-5 were 10-52% and 11-51%, respectively, where upper bounds of the ranges were obtained without batching.

To conclude, among the machine learning based methods we studied, the MLPR-TP5 network, built upon the multilayer perceptron regressor and trained with the AR model data having five discrete target values for the truncation point, was the most successful solution approach with satisfactory coverage results. This network clearly outperformed the conventional MSER and MSER-5 methods used for estimating the truncation point.

The main advantage of using MLPR for truncation point estimation is that, although human intervention would still be needed for confirmation or adjustment of the truncation point, this need would be minimal compared to MSER or the graphical method proposed by Welch (1983).

Future Research Suggestions

Studies that use simulation models and machine learning techniques together and seek solutions to the problems in the literature have increased recently. These two valuable and vital tools have successfully solved each other's problems when used together. This study contributes to the literature by successfully estimating the truncation point to eliminate the initialization bias in steady-state simulation, by means of machine learning techniques. To the best of our knowledge, although there are many studies in the literature dealing with the initialization bias problem, the number of studies using machine learning techniques for this purpose is very limited. The MLPR network has shown that it can be effective for eliminating the initialization bias by means of successful truncation point estimations. However, the LSTM and CRNN networks need to be explored further to understand the reason why they are not successful in truncation point estimation and to improve their performance in this area. The performance of these networks can be tested with different network architectures and configurations in the future.

MLPR and the other neural networks can be tested with some other more complex simulation models, particularly models of real world systems. Although the true truncation points and steady-state expected values would be unknown for those models,

expert opinion can bu used to evaluate the performance of the networks.

This study is limited to three distinct network types. Other types of neural networks with alternative network structures and configurations can be considered for solving the truncation point estimation problem. Another candidate method in this domain can be MARS (Multivariate Adaptive Regression Splines). Conceptually, the truncation point can be determined as the point in the time series at which the linear regression models fitted by MARS start having a slope of zero consistently.

Furthermore, the MLPR framework we proposed in this study can be applied to other estimation problems in stochastic simulation, such as determination of input probability distributions and statistical output analysis.

REFERENCES

[1] Afolalu, S. A., Babaremu, K. O., Ongbali, S. O., Abioye, A. A., Abdulkareem, A., Adejuyigbe, S. B. (2019, December). Overview impact of application of queuing theory model on productivity performance in a banking sector. In Journal of Physics: Conference Series (Vol. 1378, No. 3, p. 032033). IOP Publishing.

[2] Agarap, A. F. (2018). Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375.

[3] Banks, J., Carson, J. S., Nelson, B. L., Nicol, D. M. (2005). Discrete-event system simulation. Pearson Prentice Hall

[4] Bischak, D.P., Kelton, W.D., Pollock, S.M., 1993. Weighted batch means for confidence intervals in steady-state simulations. Management Science 39, 1002–1019.

[5] Cash, C.R., D.G. Dippold, J.M. Long, B.L. Nelson, and W.P. Pollard. 1992. Evaluation of tests for initial conditions bias. In Proceedings of the 1992 Winter Simulation Conference, ed., J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson, 577-585 Institute of Electrical and Electronics Engineers, Piscataway, NJ

[6] Chen, L., Li, R., Liu, Y., Zhang, R., Woodbridge, D. M. K. (2017, August). Machine learning-based product recommendation using Apache Spark. In 2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)(pp. 1-6). IEEE.

[7] Chinyere, O. U., Francisca, O. O., Amano, O. E. (2011). Design and simulation of an intelligent traffic control system. International journal of advances in engineering technology, 1(5), 47.

[8] Currie, C. S., Cheng, R. C. (2016, December). A practical introduction to analysis of simulation output data. In 2016 Winter Simulation Conference (WSC) (pp. 118-132). IEEE.

[9] Derbentsev, V., Matviychuk, A., Datsenko, N., Bezkorovainyi, V., Azaryan, A. (2020, October). Machine learning approaches for financial time series forecasting. CEUR Workshop Proceedings.

[10] Deshpande, R. R. (2012). On the rainfall time series prediction using multilayer perceptron artificial neural network. Int. J. Emerg. Technol. Adv. Eng, 2(1), 2250-2459.

[11] Fishman, G.S., 1978. Principles of Discrete Event Simulation. John Wiley, New York.

[12] Fischer, T., Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. European journal of operational research, 270(2), 654-669.

[13] Gardner, M. W., Dorling, S. R. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric environment, 32(14-15), 2627-2636.

[14] Gers, F. A., Schmidhuber, J., Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. Neural Computation, 12(10), 2451–2471. https://doi.org/10.1162/089976600300015015

[15] Giabbanelli, P. J. (2019). Solving challenges at the interface of simulation and Big Data Using Machine Learning. 2019 Winter Simulation Conference (WSC). https://doi.org/10.1109/wsc40007.2019.9004755

[16] Glynn, P. W., Iglehart, D. L. (1988). Simulation methods for queues: An overview. Queueing systems, 3, 221-255.

[17] Grassmann, W. K. (2008). Warm-up periods in simulation can be detrimental. Probability in the Engineering and Informational Sciences, 22(3), 415-429.

[18] Hijry, H., Olawoyin, R. (2021). Predicting patient waiting time in the queue system using deep learning algorithms in the emergency room. International

Journal of Industrial Engineering and Operations Management, 03(01), 33–45. https://doi.org/10.46254/j.ieom.20210103

[19] Hoad, K., Robinson, S., Davies, R. (2008). Automating warm-up length estimation. 2008 Winter Simulation Conference. https://doi.org/10.1109/wsc.2008.4736110

[20] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: S.C. Kremer, J.F. Kolen (eds.) A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press (2001)

[21] Hochreiter, S. Schmidhuber, J. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

[22] Jain, S., Narayanan, A., Lee, Y.-T. T. (2018). Comparison of data analytics approaches using simulation. 2018 Winter Simulation Conference (WSC). https://doi.org/10.1109/wsc.2018.8632330

[23] James, G., Witten, D., Hastie, T., Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: springer.

[24] Kelton, W. D., and A. M. Law. 1983. A new approach for dealing with the startup problem in discrete event simulation. Naval Research Logistics Quarterly 30 641-658.

[25] Kendall, D. G. (1953). "Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain". The Annals of Mathematical Statistics. 24 (3): 338–354.

[26] Keshtgary, M., Mohammadi, R., Mahmoudi, M., Mansouri, M. R. (2012). Energy consumption estimation in cluster based underwater wireless sensor networks using m/m/1 queuing model. International Journal of Computer Applications, 43(24), 6-10.

[27] Khalid, A., Sundararajan, A., Acharya, I., Sarwat, A. I. (2019). Prediction of li-ion battery state of charge using multilayer perceptron and long short-term memory models. 2019 IEEE Transportation Electrification Conference and Expo (ITEC). https://doi.org/10.1109/itec.2019.8790533

[28] Kim, Y. H., Jun, K. W., Joo, H., Han, C., Song, I. K. (2009). A simulation study on gas-to-liquid (natural gas to Fischer–Tropsch synthetic fuel) process optimization. Chemical Engineering Journal, 155(1-2), 427-432.

[29] Kim, K. B., Kwon, H. H., Han, D. (2018). Exploration of warm-up period in conceptual hydrological modelling. Journal of Hydrology, 556, 194-210.

[30] Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[31] Kolahi, S. S. (2011). Simulation model, warm-up period, and simulation length of Cellular Systems. 2011 Second International Conference on Intelligent Systems, Modelling and Simulation. https://doi.org/10.1109/isms.2011.63

[32] Kotsias, P. C., Arús-Pous, J., Chen, H., Engkvist, O., Tyrchan, C., Bjerrum, E. J. (2019). Direct steering of de novo molecular generation using descriptor conditional recurrent neural networks (cRNNs).

[33] Kourou, K., Exarchos, T. P., Exarchos, K. P., Karamouzis, M. V., Fotiadis, D. I. (2015). Machine learning applications in cancer prognosis and prediction. Computational and structural biotechnology journal, 13, 8-17.

[34] Kumar, N., Jha, G. K. (2013). A time series ann approach for weather forecasting. Int J Control Theory Comput Model (IJCTCM), 3(1), 19-25.

[35] Kyritsis, A. I., Deriaz, M. (2019). A machine learning approach to waiting time prediction in queueing scenarios. 2019 Second International Conference on Artificial Intelligence for Industries (AI4I). https://doi.org/10.1109/ai4i46381.2019.00013

[36] Laidler, G., Morgan, L. E., Nelson, B. L., Pavlidis, N. G. (2020). Metric Learning for Simulation Analytics. 2020 Winter Simulation Conference (WSC). https://doi.org/10.1109/wsc48552.2020.9383904

[37] Law, A. M. (2015). Simulation modeling and analysis (5th ed.). McGraw-Hill Education.

[38] Linton, J. R., Catherine. (n.d.). A comparison of selective initialization

bias elimination methods. Proceedings of the Winter Simulation Conference. https://doi.org/10.1109/wsc.2002.1166495

[39] Li, M. P., Sankaran, P., Kuhl, M. E., Ganguly, A., Kwasinski, A., Ptucha, R. (2018, December). Simulation analysis of a deep reinforcement learning approach for task selection by autonomous material handling vehicles. In 2018 Winter simulation conference (WSC) (pp. 1073-1083). IEEE.

[40] Le, X. H., Ho, H. V., Lee, G., Jung, S. (2019). Application of long short-term memory (LSTM) neural network for flood forecasting. Water, 11(7), 1387. https://doi.org/10.3390/w11071387

[41] Lee, Y. H., Kyung, K. H., Jung, C. S. (1997). On-line determination of steady state in simulation outputs. Computers industrial engineering, 33(3-4), 805-808.

[42] Little, J. D. (1961). A proof for the queuing formula: l = w. Operations Research, 9(3), 383–387. https://doi.org/10.1287/opre.9.3.383

[43] Liu, Y., Yan, L., Liu, S., Jiang, T., Zhang, F., Wang, Y., Wu, S. (2020, December). Enhancing input parameter estimation by machine learning for the simulation of large-scale logistics networks. In 2020 Winter Simulation Conference (WSC) (pp. 608-619). IEEE.

[44] Mahajan, P. S., Ingalls, R. G. (2004, December). Evaluation of methods used to detect warm-up period in steady state simulation. In Proceedings of the 2004 Winter Simulation Conference, 2004. (Vol. 1). IEEE.

[45] Modi, M., Agarwal, G., Patil, V., Khare, A., Shukla, S., Sankhala, A. (2019). Minimization of traffic congestion by using queuing theory. International Journal Of Scientific and Technology Research, 8(10).

[46] Mohapatra, S., Yang, T., Gómez-Bombarelli, R. (2020). Reusability report: Designing organic photoelectronic molecules with descriptor conditional recurrent neural networks. Nature Machine Intelligence, 2(12), 749-752.

[47] Mokashi, A. C., Tejada, J. J., Yousefi, S., Tafazzoli, A., Xu, T., Wilson, J. R., Steiger, N. M. (2010, December). Performance comparison of MSER-5 and N-Skart on the simulation start-up problem. In Proceedings of the 2010 Winter Simulation Conference (pp. 971-982). IEEE.

[48] Nelson, D. M., Pereira, A. C., de Oliveira, R. A. (2017). Stock market's price movement prediction with LSTM neural networks. 2017 International Joint Conference on Neural Networks (IJCNN). https://doi.org/10.1109/ijcnn.2017.7966019

[49] Nguyen, H. D., Tran, K. P., Thomassey, S., Hamad, M. (2021). Forecasting and Anomaly Detection approaches using LSTM and LSTM Autoencoder techniques with the applications in supply chain management. International Journal of Information Management, 57, 102282.

[50] Noriega, L. (2005). Multilayer perceptron tutorial. School of Computing. Staffordshire University, 4(5), 444.

[51] Oh, H. S., and Park, K. J. (2015). An effective heuristic for initial bias reduction in simulation output. ASIA LIFE SCIENCES, 12, 265-275.

[52] Orriols-Puig, A., Casillas, J., Bernadó-Mansilla, E. (2008). Genetic-based machine learning systems are competitive for pattern recognition. Evolutionary Intelligence, 1, 209-232.

[53] Predescu, E. F., Tefan, A., Zaharia, A. V. (2019). Software effort estimation using multilayer perceptron and long short term memory. Informatica Economica, 23(2), 76-87.

[54] Remy, P. (2020). Conditional RNN for Keras. GitHub repository. Retrieved from https://github.com/philipperemy/condrnn

[55] Robinson, S. (2002, December). A statistical process control approach for estimating the warm-up period. In Proceedings of the Winter Simulation Conference (Vol. 1, pp. 439-446). IEEE.

[56] Robinson, S., Ioannou, A. (2007). The problem of the initial transient: Techniques for estimating the warm-up period for discrete-event simulation models. Warwick Business School, University of Warwick, Coventry, UK, 1-30.

[57] Roth, E., 1994. The relaxation time heuristic for the initial transient problem in M/M/k queueing systems. European Journal of Operational Research 72:376-386.

[58] Sagheer, A., Kotb, M. (2019). Time series forecasting of petroleum production using deep LSTM recurrent networks. Neurocomputing, 323, 203-213.

[59] Sanchez, P. J., White, K. P. (2011). Interval estimation using replication/deletion and MSER truncation. Proceedings of the 2011 Winter Simulation Conference (WSC). https://doi.org/10.1109/wsc.2011.6147778

[60] Sandıkcı, B., Sabuncuoglu, İ. (2006). Analysis of the behavior of the transient period in non-terminating simulations. European Journal of Operational Research, 173(1), 252-267.

[61] Schriber, T.J., Andrews, R.W., 1984. ARMA-based confidence intervals for simulation output analysis. American Journal of Mathematical and Management Sciences 4, 345–373.

[62] Schruben, L.W. 1982. Detecting initialization bias in simulation output. Operations Research, 30(3):151-153.

[63] Sherzer, E., Senderovich, A., Baron, O., Krass, D. (2022). Can machines solve general queueing systems?. arXiv preprint arXiv:2202.01729.

[64] Shen, S., Jiang, H., Zhang, T. (2012). Stock market forecasting using machine learning algorithms. Department of Electrical Engineering, Stanford University, Stanford, CA, 1-5.

[65] Singh, V. K., Pandey, S. (2016, March). Minimum configuration MLP for solving XOR problem. In 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 174-179). IEEE.

[66] Soutner, D., Müller, L. (2013). Application of LSTM neural networks in language modelling. In Text, Speech, and Dialogue: 16th International Conference, TSD 2013, Pilsen, Czech Republic, September 1-5, 2013. Proceedings 16 (pp. 105-112). Springer Berlin Heidelberg.

[67] Spratt, S. C. (1998). Heuristics for the startup problem. Department of Systems Engineering, University of Virginia.

[68] Sundari, M.S., Palaniammal, S. (2015). Simulation of M / M / 1 Queuing System Using ANN.

[69] Teresa, C. O. M., Paegelow, M., Mas, J.-F., Escobar, F. (2018). Geomatic approaches for modeling land change scenarios. Springer International Publishing.

[70] Van Houdt, G., Mosquera, C., Nápoles, G. (2020). A review on the long short-term memory model. Artificial Intelligence Review, 53, 5929-5955.

[71] Wang, W., Wong, A.K. (2002). Autoregressive Model-Based Gear Fault Diagnosis. Journal of Vibration and Acoustics, 124, 172-179

[72] Welch, P. D. 1983. The statistical analysis of simulation results. In Computer Performance Modeling Handbook, ed. Lavenberg, 267-329.

[73] Welch, P.D. 1983. The statistical analysis of simulation results. In The computer performance modeling handbook, ed. S.S. Lavenberg, pp. 268–328. New York: Academic Press.

[74] White, K. P., Jr.. 1997. An effective truncation heuristic for bias reduction in simulation output. Simulation,

[75] White, K. P., Cobb, M. J., Spratt, S. C. (n.d.). A comparison of five steady-state truncation heuristics for Simulation. 2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165). https://doi.org/10.1109/wsc.2000.899843

[76] Yulita, I. N., Abdullah, A. S., Helen, A., Hadi, S., Sholahuddin, A., Rejito, J. (2021). Comparison multi-layer perceptron and linear regression for time series prediction of novel coronavirus covid-19 data in West Java. In Journal of Physics: Conference Series (Vol. 1722, No. 1, p. 012021). IOP Publishing.

[77] Zeiler, M. D., Ranzato, M., Monga, R., Mao, M., Yang, K., Le, Q. V., ... Hinton, G. E. (2013, May). On rectified linear units for speech processing. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 3517-3521). IEEE.