Lakehead University

Knowledge Commons,http://knowledgecommons.lakeheadu.ca

Electronic Theses and Dissertations | Electronic Theses and Dissertations from 2009

2020

Development of simulation and machine learning solutions for social issues

Fisher, Andrew

http://knowledgecommons.lakeheadu.ca/handle/2453/4743

Downloaded from Lakehead University, KnowledgeCommons

Development of Simulation and Machine Learning Solutions for Social Issues

by

Andrew Fisher

A thesis submitted in partial fulfillment of the

requirements for the degree of

Master of Science

in

Computer Science

in the

Faculty of Science and Environmental Studies

of

Lakehead University, Thunder Bay

Committee in charge:

Dr. Vijay Mago (Principal Supervisor)
Dr. Eric Latimer (Co-Supervisor)
Dr. Philippe Giabbanelli (Co-Supervisor)

Dr. Thangarajah Akilan (External Examiner)
Dr. Zubair Fadlullah (Internal Examiner)

Fall 2020

The thesis of Andrew Fisher, titled Development of Simulation and Machine Learning Solutions for Social Issues, is approved:

Chair _____ Date _____

_____ Date _____

_____ Date _____

Lakehead University, Thunder Bay

Development of Simulation and Machine Learning Solutions for Social Issues

Copyright 2020

by

Andrew Fisher

Abstract

DEVELOPMENT OF SIMULATION AND MACHINE LEARNING SOLUTIONS FOR SOCIAL ISSUES

When developing solutions for social issues, it can be difficult to evaluate the impact they may have without a real world implementation. This may not be possible for reasons such as resource, time, and monetary constraints. To resolve these issues, simulation and machine learning models can be used to mimic reality and provide a picture of how these solutions would fare. In Chapters 3 and 4, a deep learning approach to simulating homelessness populations in Canada is presented. This model would provide policy makers with a tool to test different solutions for this societal problem without the need to wait for approvals or funding from local officials. In addition to this solution, data enhancement techniques are presented as a comprehensive dataset on homeless population transitions for such a model to learn from does not exist. Lastly, Chapter 5 presents a transfer learning architecture to detect tents in satellite images. The motivation for this work was that "tent camps" are common for homeless populations to live in and by having a solution to detect these from images, policy makers can easily see where to focus resources such as shelters for example. Similar to the constraint present with the homelessness simulation, a comprehensive dataset on tents in satellite images does not exists. Therefore, this chapter also presents a solution to generate an comprehensive dataset for the architecture to learn from. The result of this thesis is developed solutions to social issues that utilize the power of machine learning and simulation models.

Dedication

This is dedicated to my family and friends for their endless support, my high-school computer science teacher Mr. Matteo Di Muro for igniting my interest in the field, and the educators throughout my academic journey who believed in me every step of the way.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank Dr. Abhi Rao for helping proof-read the work in Chapter 3. For their invaluable insights and expertise, I would like to thank my supervisors as well as Dr. Tim Aubry, Dr. Bartosz Gajderowicz, Dr. Emad Mohammed, and Dr. Thangarajah Akilan. For supporting me in scholarship applications and answering my endless questions, I would like to thank Andrew Heppner and Maegen Lavallee. I also extend my thanks and gratitude to Dr. Robert Bryce as well as all of my colleagues at the DaTALab in the CASES building at Lakehead University.

For providing support throughout my degree, I am immensely thankful for the Insight grant 21820 from the Social Sciences and Humanities Research Council (SSHRC), as well as the resources provided to me from the DaTALab. Additionally, the costs of publication and other expenses were covered by a NSERC Discovery Grant held by my supervisor, Dr. Vijay Mago.

Chapter 1

Introduction

This thesis mainly comprises of a collection of articles produced over the course of this degree. The main focus throughout these works was to solve social related issues using simulation and machine learning techniques. In Chapter 2, the thesis will provide a background on relevant topics in this domain that are necessary to understand for the articles to follow. The topics will include the basis of simulations (Chapter 2.1), discrete models (Chapter 2.2), numerical methods (Chapter 2.3), and machine learning methods (Chapter 2.4).

In Chapter 3, this thesis will present an article published in IEEE Access about a deep learning model created to simulate homeless populations [35]. It is estimated that over 235,000 Canadians experience homelessness at some point each year [19, 23, 78, 124, 125, 129]. With the emergence of smart cities, it would be beneficial to leverage the processing power of deep learning to assist in the planning and testing of different policies to address this issue. When examining a population of homeless individuals, one can view them as being distributed, at any one point in time, among several possible states: for example, the street or an emergency shelter. This chapter aims to provide a means of simulating across these states, including no longer homeless, over time. The probability that an individual will transition

from one state to another is called a transition probability. Thus, by creating a matrix of transition probabilities between all of the states, one would have a transition probability matrix. If the problem was simply approached by using a mathematical model such as a Markov decision process, one would run into the issue of how to accurately adjust the probabilities to produce realistic results. Ideally, there would be a model that can reasonably modify them based on real-life data. To do this, the chapter introduces two modified deep learning algorithms; modified deep q-learning (MDQL) and modified neural fitted q-iteration (MNFQ). These algorithms dynamically produce a set of transition probability matrices for each week of the year. The modifications made to these algorithms to adapt to the homelessness problem are discussed that were necessary to create the simulation model. After training it on high resolution, weekly data, the results will show that when running it on a low resolution data set that spans 3 years, the model is able to achieve a relative percent difference from the final population of 12.5%. Immediately following in Chapter 4, this thesis will present an extension of this work that is currently being finalized for submission in a journal.

Lastly, in Chapter 5, the thesis presents a deep learning algorithm published at IEEE Systems, Man, and Cybernetics (SMC) that can be used detect whether or not a tent is in a satellite image. The motivation for this work was to provide a novel contribution to the problem of detecting tents in satellite images, to the authors' knowledge, it has only seen mathematical approaches in the past [131, 117]. Additionally, because homeless populations often setup tents for shelters, an algorithm such as the one proposed would help policy makers easily determine where appropriate services should be allocated. This approach uses three deep learning methods that utilize transfer learning from the ResNetV2, InceptionV3, and MobileNetV2 models, trained on ImageNet, attached to a unique architecture referred to as "TentNet". The performance of these models is first shown in detecting planes and ships

within satellite imagery in previously defined datasets as a baseline. Then, a new dataset is created from a compilation of tents from the xView project to use for testing, along with another dataset of synthetic images from the generative adversarial networks StyleGAN2 and DCGAN for training. After training on a dataset containing only synthetic images for the tents class, the ResNetV2 architecture achieved the highest accuracy of 73.68% when testing on the real satellite imagery.

Chapter 2

Background

All of this chapter will be submitted as the following peer-reviewed journal article:

- Fisher, A., Giabbanelli, P., & Mago, V. (2021). Impact of Machine Learning on Simulation Systems.

Over the course of my degree, I researched topics related to simulation systems to expand my knowledge in the field. As a result, I have created summarizations of many related works that fit into the background section of this thesis. Using the text here and more research to come, we plan to publish an extensive survey article.

2.1 Simulations

A *system* can be defined as a collection of parts or processes that are organized for some general purpose [20]. There are four main classes of systems [14, 17]:

(1) Natural systems: a system whose origins are that of the universe

(2) Designed physical systems: physical systems from human design

(3) Designed abstract systems: abstract systems from human design

(4) Human activity systems: an ordered system that consists of human activity

Improving a *real-life* operations system through experimentation can be a costly, time consuming, and potentially unethical process. Indeed, a physical experiment can take time to design, months to implement, and its effect may have to be measured after years. The experiment can be expensive, for instance when conducting large randomized-controlled trials or changing urban components in a city (e.g., creating new shelters as an experiment to reduce homeless). Experiments may be straightforward: to understand the effect of missing treatment onto viral load in a human body, the best observations would be obtained when preventing subjects from getting treatment. However, denying treatment would be unethical, thus such experiments are not feasible. Any one of the these three reasons can lead to the use of simulated experiments (or '*simulations*') instead of physical experiments. Simulations can be defined as a simplified imitation of a system, on a computing device, that can be used to perform experimentation for the purpose of better understanding and/or improving the system [108]. In the context of this thesis, there are two main types of simulation [25]:

- Equation-based: where the population (structure) of the system can be described by a variable (e.g., number). This is also know as a macro-level or (quantitative) aggregate model. Typical techniques include Systems Dynamics or Fuzzy Cognitive Maps.

- Individual-based: where the entities in the simulation are either individuals (agents) with decision-making capabilities or are simply objects (resources). This is also known as a micro-level or individual-oriented model. Typical techniques include Agent-Based Modelling, Cellular Automata, or Network models.

When considering a simulation approach in place of physical experimentation, one needs to consider the overall *goal* of developing such a model. Reasons could include predicting outputs, better understanding a system, discovery of new uncertainties, and so forth. The valuable information received from a simulation model comes from the capture of qualitative behaviors [29]. It provides a methodical approach to assist in understanding the key aspects of a system and stimulate group communication about how the mechanisms of a system [4]. With the use of computational models, understanding, designing, managing and predicting the workings of complex systems and processes is a simple feat once the model has been created [14].

The process of developing a model and executing it for simulations is summarized in Figure 2.1, based on the descriptions from [4, 5, 9]. Once the final step has been completed, an *operational phase* takes place where the model is verified against the steps previously stated then experimented with to fine tune the quality of the outputs. This verification phase introduces a loop in the process since the fine tuning may involve a change in the conceptual model, simulation model, or program.

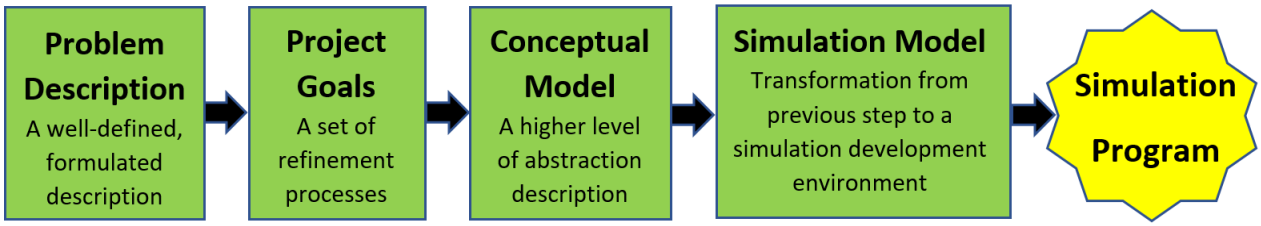


Figure 2.1: A process for creating a simulation program based on descriptions from [4, 5, 9].

2.2 Discrete Models

Network Model

A network model *generalizes* the notion of a cellular automaton as any arbitrary topology can now be used instead of a regular tesselation such as a 2D grid. The entities (i.e., nodes) in a network model can also have a richer set of states, using variables such as age and income, instead of carrying only one categorical state. An example of this discrete model was presented in [45] where it aimed to "illuminate the core dynamics [29] of social influences and binge drinking using an agent-based social network [47]". In this work, they ran simulations based on hypotheses regarding different ways individuals could be connected based on independent variables. This was achieved using two consecutive rules applied to the network [45]: *selection* rules specifying the types of persons that an individual would befriend (e.g., using similarity between individual features to infer the existence of a social tie); and *influence* rules to specify the effect of peers on an individual (e.g., following the behavior of most peers in a 'voting' approach).

For selection rules, the authors described two different hypotheses: random (assumes that an individual is equally likely to connect to any of the other peers in the population) and

similarity (probability that two individuals will connect is based on the number of features in common). For influence rules, they also compared two approaches: majority (individual is assigned the state most taken on by their peers) and fractional (individual is assigned a state taken on by a critical fraction of their peers) [45].

Regardless of the specific instantiation of rules, a network model can be conceptualized as connecting (or 'networking') an individual to others in order to observe the impact that peers will have.

Agent-Based Model

Agent-Based Models (ABMs) generalize the notion of network model: in addition to the use of a social network to specify the patterns and function of social interactions, there can be a *physical* network enabling interactions between agents and the environment. An example of this discrete model was presented by Khademi *et al.* [70] where it looked to use an agent-based model (ABM) to capture changes in social norms that affect eating behaviors. An ABM can be defined as a predictive model that seeks to simulate what may happen in the future. Each of the entities in the population of the model are explicitly represented such that they interact with each other and/or their environment through discrete time steps [4, 70].

In the case of Khademi *et al.*, they used a network model to generate the interaction patterns between the agents [46]. The network was specifically *small-world*, which means that (*i*) connections between individuals are, on average, only a short distance away from each other; and (*ii*) individuals tend to form communities. To see how changes in social norms affect the population, a *virtual intervention* is introduced to the model so that a certain number of individuals are influenced to change their state. From this, the population

is simulated to see how/if that changed population influences the neighboring individuals [70]. These types of models will be further explored in Chapters 3 and 4.

2.3 Numerical Methods

Overview of methods

The problem of interpolating irregularly-spaced data points has been studied for many decades. Consider four methods that will be briefly introduced: Radial Basis Function (RBF) [103], Inverse Distance Weighting (IDW) [116], Quadratic Polynomials (QP) [39], and Least Squares (LS) [39]. As presented in [103], RBF represents the interpolating function as a linear combination of basis functions, one for each training point. The basis functions, in this case, only depend on the *distance* (or difference) from the prediction point to the training point. In [116], IDW is described as an interpolating method where the unknown points (test points) are calculated with the weighted average of the sampling points (training points). For QP, as the name suggests, it is simply a quadratic polynomial function that best fits the training data presented to it. Lastly, for LS, it fits a linear model with coefficients to minimize the sum of squares between the training data in the dataset and the output from the linear approximation.

A more modern and commonly used method is called *kriging* [112]. It is widely used because it is fast to train and is generally more accurate than other types of surrogate models. However, the prediction time of kriging increases with the size of the dataset, and the training can fail if the dataset is too large or poorly spaced, which limits the accuracy that is attainable" [60]. It is an interpolating method that combines a known function (such as Gaussian correlation) with a stochastic process in a linear manner. This results in a

function that uses linear regression coefficients to produce accurate outputs as it doesn't perfectly follow that of a smoothed spline along the data but rather, it follows the given data points perfectly.

Variations of kriging have been introduced to further improve the results and resolve some of the issues noted before. KPLSK [12] extends kriging to have a partial least squares (PLS) regression approach. This is suitable for higher dimensional problems and uses fewer hyper-parameters to still achieve a high accuracy. GE-KPLS [10] extends both kriging and KPLSK models by adding gradient enhancing. This involves exploiting gradient information via a slight increase in the size of the correlation matrix to result in a reduction of the number of hyper-parameters. It *typically* produces a higher accuracy than kriging but is not as computationally efficient. RMTS [60] is described as being effective on low-dimensional problems with a large number of instances. It works by computing coefficients of splines that are trying to fit the resulting output as best as possible. This model has two choices when picking the spline:

(1) RMTB: This uses a B-spline and is a better choice when *training time* is an important factor

(2) RMTC: This uses a cubic hermite and is a better choice when *interpolation accuracy* is an important factor

A Guiding Example to Contrast Numerical Interpolation Methods With the SUMO Toolbox

Consider a basic simulation model for forest fires that is represented as a grid of cells. Assume that a cell in this model can be in one of three states at a given time: empty (0), tree (1),

or fire (2). For each time stamp, each cell in the grid is examined. If it is a tree, it will see if there is currently (before any other square was updated this time stamp) a fire to the left, right, above, or below of it; that is, it considers a *Von Neumann* neighborhood configuration. If any of those surrounding cells are indeed on fire, the tree being considered starts on fire as well. However, if the cell is a fire, it simply becomes an empty square. The grid these cells are in has a *closed* boundary where the edges are the end of the forest. The forest is initialized with the parameters grid size N and tree probability (percent chance a tree will spawn in each grid square) P These are used in the following process:

(1) Create a grid of size $N \times N$

(2) Cycle through each square in the grid and with a probability of P, populate it with a tree

(3) For each tree in the first column of the grid, start it on fire

The simulation is complete when there are either no trees left or no fires left. This model was with $n \in \{100, 500, 700, 1000\}$ and $p \in [20, 69], p \in \mathbb{N}$, replicated 5 times. In total, there are $3 \times 50 \times 5 = 1000$. For each run, the total percentile of the forest burned once the simulation completed was recorded as the class.

With this data, the goal is to apply a surrogate model to it that can accurately predict the total percentile of the forest burned, based on (1) grid size and (2) tree probability. To achieve this, the surrogate Modeling Toolbox (SMT) [11] was used to try the following models: least squares (LS), inverse distance weighing (IDW), quadratic polynomials (QP), radial basis function (RBF), Regularized Minimal-energy Tensor-product B-splines (RMTB), and Regularized Minimal-energy Tensor-product Cubic hermite splines (RMTC). Models not used from this library include: Kriging, KPLSK, and GEKPLS.

Consider Figure 2.2 which shows how each of the used models performed on this dataset. A *perfect* model would have all of the red points along to line to show that it was accurately able to predict the class (ie. percentage of trees burned). To further visualize these results, consider Figure 2.3. It first shows the Root-Mean-Squared-Error (RMSE) of the model on the given data set (ie. how it performed when testing). Then, it shows the predicted values for three, larger [grid size, percentage] combinations.

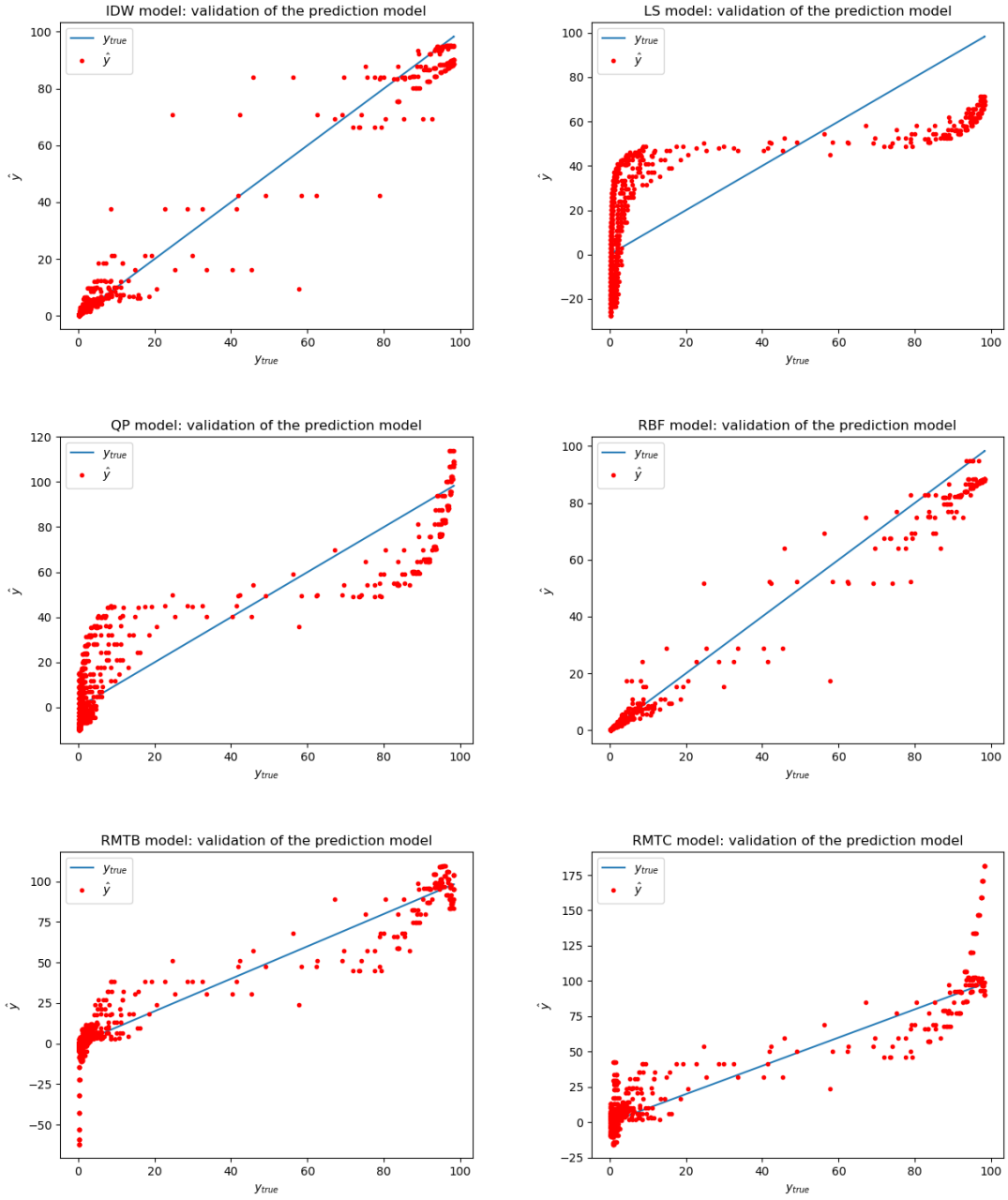


Figure 2.2: SMT models' results when attempting to predict the number of trees burned in a basic forest fire simulation model.

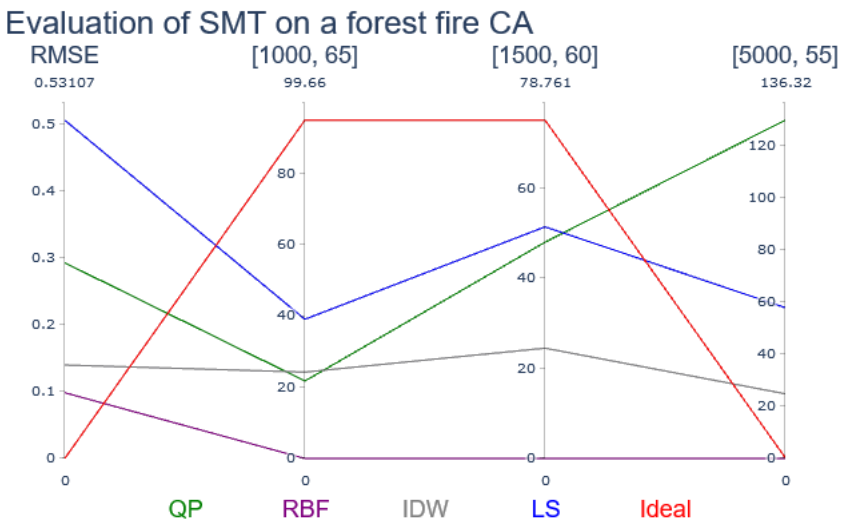


Figure 2.3: A prediction visualization from the trained models in Figure 2.2 that shows how they did when extrapolating.

The code shown in Box 1 would be used to validate a model where the *training* and *testing* values are pulled from the dataset noted before. The outcome of experimenting with this library was that these models are more for *interpolation* rather than *extrapolation*. This is seen from the low RMSE and, for a majority of the models, an inaccurate prediction respectively. Indeed, in the original paper [11], the models used were described as "interpolating method(s)"; the best *interpolating* model for this simulation was *RBF*.

For the predictions, these particular points were ran to retrieve the results but the models were not trained on them to see how close they were. The creation of figure 2.3 did show that the *LS* model did follow the closest to the *Ideal* plot for *extrapolation* but it was still, however, quite off.

2.4 Machine Learning Methods

Process Overview

When using a machine learning algorithm, one must consider how the input data is selected and processed. Consider the simplified set of steps in Figure 2.4.

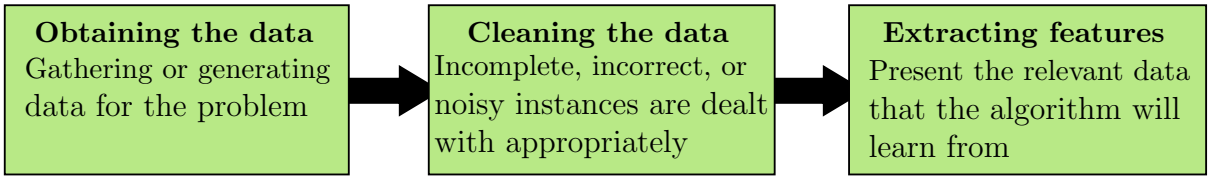


Figure 2.4: Simplified steps for data processing in machine learning to ensure that the algorithm has a clean dataset to work with.

Once these generalized steps have been completed, the data is ready to be presented to the algorithm. Consider the following basic steps:

(1) Select an algorithm to work with

- These can include, but are not limited to, the discrete models discussed in section 2.2

(2) Experiment with different parameters

- For example, the "learning rate" for neural networks in section 2.4 could be changed to see how it affects the results

(3) If applicable, simplify the data set or present different features to the algorithm

- Similar to the previous step, the goal here is to see how this affects the output

Once the model has been fine tuned, the next process is to evaluate the outputs. A common approach is a method called "10-fold cross-validation" and is described in Figure 2.5. This process will be used in Chapter 5 to fairly evaluate the different models.

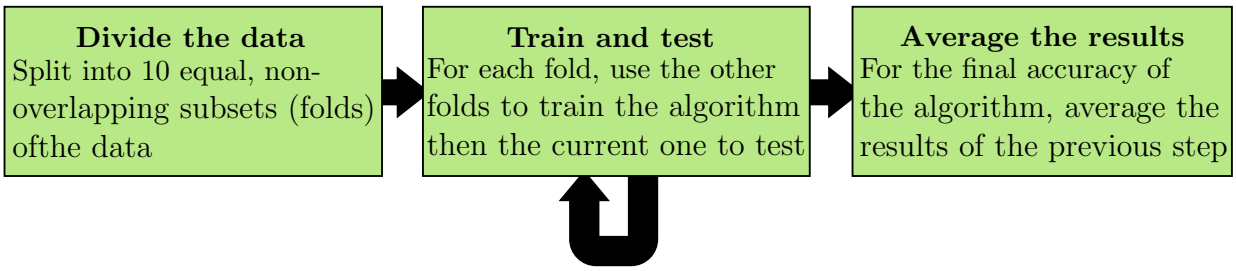


Figure 2.5: An overview of the process for 10-fold cross-validation which is an accurate and fair way to evaluate machine learning models on a given dataset.

Surface Temperature (C^o)	Atmosphere	Habitable
14.9	Thin	True
-63	Very Thin	False
-20	Thick	True
11	Thin	True
-145	Very Thick	False

Table 2.1: Example dataset with two features and one binary class that an algorithm could learn to attempt to interpolate or extrapolate new datapoints.

OneR

Consider the sample data in Table 2.1 that describes different planets. In this set, it provides two *features* (surface temperature and atmosphere) and a *classification* (whether or not it's habitable). If one were to add another planet to this list but didn't know whether or not it was habitable, they may look at the previous data in the set to determine it. One way

to do this could be to generate a set of *rules* and see which one the new data follows. For example, consider the following rule:

```
if (Atmosphere == Thin or Atmosphere == Thick) then
Habitable = True
```

This rule is filtering based on **one** feature- which is exactly what the OneR algorithm sets out to do. It simply generates a rule on a single feature that classifies a majority of the data.

Decision Tree

The rule generated by OneR could be further refined by using the *surface temperature* feature to get the following rule:

```
if (Atmosphere == Thin or Atmosphere == Thick) and
(Surface_Temperature >= -20) then Habitable = True
```

This forms the basis of the concept of *decision trees* [56]. In the context of a rule, an instance of data would start at the beginning of each if-statement (root of the decision tree), see which of the *first* conditions it met (which *branch* in the tree from the root node) then, if applicable, see which of the second condition(s) it met (after travelling down the first branch, which of the next branches does it satisfy), and so forth until it reaches the classification (the *leaf* or bottom-most node in the tree). The data set, however, may not always *perfectly*

satisfy the rules. In this case, the classification that gives the smallest error is chosen for the rule in question [83].

Random Forest

For a random forest, as the name implies, it creates a *set* of decision trees by looking at the data set randomly. That is, random instances (rows) of the data set and random features (columns) to build the trees within that. Once a specified number of trees have been built, the ones with the lowest errors are combined into a resulting output tree that consists of many *sub-trees* [56]. The advantage with this method as opposed to simply creating decision trees is that it helps avoid *over-fitting* the data [83]. This becomes a problem when new data is introduced as it may not fit the patterns present in the initial data set.

Support Vector Machines

Consider the sample data set in Table 2.1 as represented in Figure 2.6.

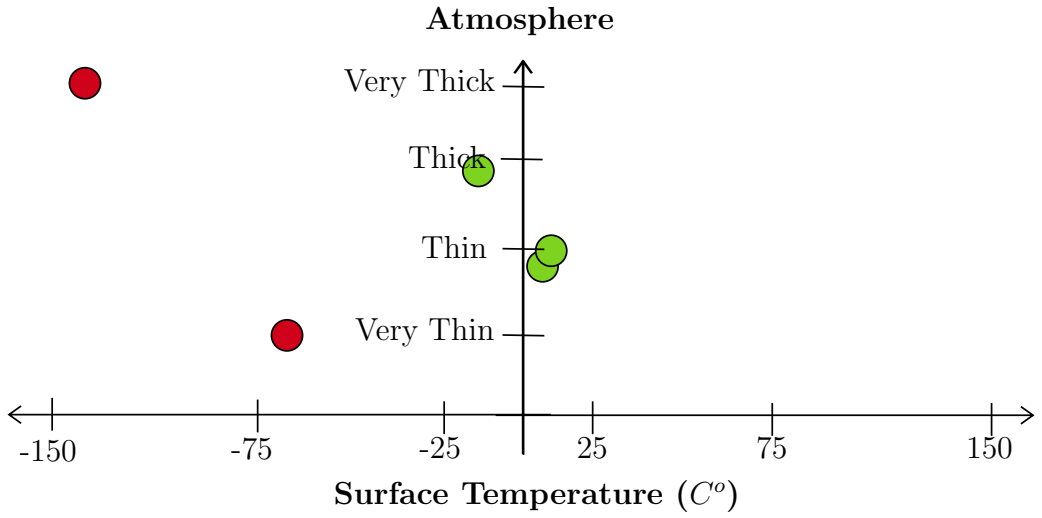


Figure 2.6: Table 2.1 represented as a graph to visually see how it would be plotted.

Where green is a habitable planet (true) and red is not (false). If one were to try and determine if a new data point on the graph is habitable or not based on the given data, imagine a line in Figure 2.6 that best separates the green points from the red ones. Then, for the new data, one would see which side of the line it fell on (ie. habitable or not) and classify it that way. This is the core concept of *support vector machines* [16] [114]. The way that this algorithm determines the best "separation line" (or hyper-plane [53]) for the data is based on each of the closest data points' *distance* from the line. These points are known as *support vectors* and the goal is to produce the minimal distance for each one of them [53] [114].

This example shows a two-dimensional plane as that's how many *features* the sample data set in Table 2.1 has. However, it is possible to produce a graph on a higher dimensional plane (more features) that generating a hyper-plane for may be beneficial [53]. The same concept applies here but for a very high dimensionality of features the use of *neural networks* is a common approach [53]. In the context of section 2.4, the *features* for each instance in the data set would be used as inputs to the network, with the resulting output being the distance from the hyper plane. As the network is trained with more data, the output is fine tuned to fit the optimal solution [53].

Neural Networks

This framework for machine learning, more commonly referred to as Artificial Neural Networks (ANNs), is an attempt to closely mimic that of one's cerebral cortex [53]. Figure 2.7 is a visual representation of a basic neural network.

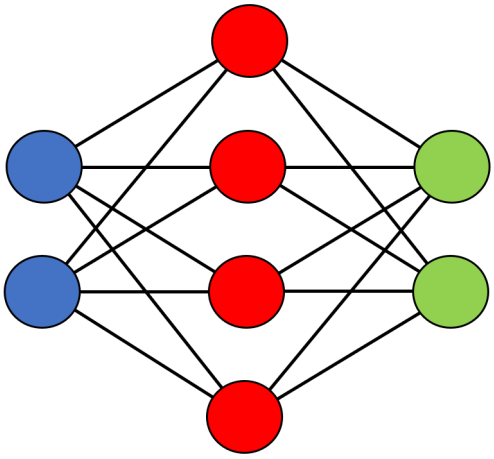


Figure 2.7: A basic example neural network where the *blue* neurons are the inputs, *red* are part of the hidden layer, and *green* are the outputs.

Each circle in Figure 2.7 represents a *neuron* in the network while each line represents a *connection* between the neurons. The blue neurons represent the *input layer*, the red neurons represent the *hidden layer*, and the green neurons represent the *output layer*. There can be more than one hidden layer, and the output layer does not necessarily need to have the same number of neurons as the input layer; Figure 2.7 is just a basic representation. Each connection has a parameter known as the *weight* of the connection and each hidden neuron, as well as output neuron, has a parameter known as its *bias* [53].

When values are presented to the input layer, each neuron has its own input value that is passed along each of its connections to the hidden layer. This value is affected by the weight of the connection it is travelling down. The output of the hidden neuron is affected by all of these values travelling to it, as well as its own bias. This is defined by the following equation:

$$O_i = \sum_j w_{ij} O_j + \theta_i \tag{2.1}$$

where O_i is the output of the i^{th} hidden node, w_{ij} is the *weight* from the j^{th} input node to the i^{th} hidden node, O_j is the output of the j^{th} input node, and θ_i is the *bias* of the i^{th} hidden node. The resulting output is then typically modified by an *activation function*. One that is commonly used is called the *sigmoid* function:

$$f(O_i) = \frac{1}{1 + e^{(-O_i)}} \tag{2.2}$$

if the network had multiple hidden layers, this process would still apply as the values travel through the network. For the example network in Figure 2.7, these outputs would then be passed to the output layer where the same process would occur to get the final output values.

To "train" the network (ie. minimize the error) a typical process known as *back propagation* is performed. Once the output layer has been calculated, the error for each neuron is calculated as follows:

$$\delta_i = (y_i - y_i') \tag{2.3}$$

where δ_i is the *error* of the i^{th} output neuron, y_i is the expected output, and y_i' is the actual output of the neuron in question. From this, the algorithm works backwards to update the weights to the output layer as follows:

$$\Delta w_{ij} = \eta \ \delta_i \ y_i' \times f(O_j) \tag{2.4}$$

where η is the *learning rate* for the network; a value used to slow (smaller value) or increase (larger value) the learning of the algorithm. This value is then added to the weight it is being calculated for. For the hidden layer's error, consider the following equation:

$$\delta_j = \sum_i \delta_i \, y_i' \, w_{ij} \tag{2.5}$$

for Figure 2.7, the same equation as 2.4 is then used to update each weight from the input layer to the hidden layer.

Q-Learning

Q-learning is a form of reinforcement learning where the model (or agent) is trying to determine an optimal action to perform while in a certain state [133]. The way it learns is by receiving a *reward*, positive or negative, for performing said action based on how *optimal* it is for an overarching goal. Consider a simple example known as the *cart-pole* problem [107]. The environment is initialized with a cart that can move left or right and an upright pole attached to it that is slightly offset such that it will fall over if the cart is not moved. The goal of the agent is to keep the pole as upright as possible. If it falls past a certain threshold (such as 15 degrees from the center for example) the agent fails, and the environment is re-initialized. The rewards given in this problem are based on how well the agent corrects the falling of the pole, that is, how close the pole stays to 0 degrees from the center [107].

In deep q-learning, the algorithm aims to learn an optimal rule or policy for a Markov decision process. It does not require a model to process the data; it simply takes a starting state, performs an action, and observes the new state reached. The learning here also comes from a reward given for reaching the new state, which the algorithm aims to maximize. When implementing this approach in a neural network, it could be defined as the *distance* from the desired output that it wants to reach.

This algorithm uses a *target network* and a *prediction network* to determine the *loss* of the system where the output of the prediction is trying to converge with the target. Every

c number of epochs, the architecture of the *target network* is updated with that of the *prediction network* being trained during this process. In order to maximize the reward, the algorithm modifies its q-values, which are used when deciding which action to perform as noted before. Given a sufficient amount of time, the algorithm can get close to converging on the desired output [28, 62, 133].

NFQ-Learning

In Neural Fitted Q-learning (NFQ), the main concept is to update the q-values in q-learning using an off-line approach where all previous experiences are considered [107]. These experiences are defined as the original state, action taken, and the resulting state. The reasoning for this is because when a q-value is updated it could, inadvertently, affect somewhere else in the table and require an update there as well.

Since this is using an off-line approach (that is, all previous experiences are considered instead of just the current experience), it will determine the best course of action for the current q-value being updated in an attempt to avoid this. This results in fast training times with a minimal amount of input [107] [139]. In addition to this, the algorithm also does not require a model to process data. It does, however, need a task to be defined which can be broken down into three points [107]:

- Avoidance control
  - A definition to keep the system (neural network) somewhere within the 'valid' region of the state space (output)
- The goal

  - As soon as the system reaches a certain area in the state space (the goal) the task is finished

- Regulator

  - Once the system reaches a certain area in the state space, a controller actively works to keep it there

This algorithm, as well as the previous two sections, are important to understand for the work presented in Chapters 3 and 4.

Transfer Learning

The learning of these metamodels is achieved through the observation of previous experiments performed where these predictive values were recorded. Once the model has been trained, it can then be used to predict the outcome of other runs simply based on the input parameters. But what happens when the experiments are too costly to perform in real life? Or when there simply isn't a significant amount of experiments for a range of input parameters to train the model? Without proper techniques, the result can be an untrustworthy model with unrealistic predictions.

To resolve this, consider the concept of transfer learning and the inertial confinement fusion (ICF) experiments as presented in [75] as an example. To summarize, these experiments required an expensive laser that was costly to run for long periods of time but did have cheaper experiments that could be done instead. The result was a simulation model for the cheap experiments and a few instances for the expensive experiments [75]. With the simulation model, the authors created a dataset containing 1,000 instances. Since the cheaper experiments were ran so much that an accurate simulation could be created, there's no desire

to create a surrogate model that can predict these outputs nor resource requirements. But the knowledge from these runs is useful as the outputs may contain a pattern that the model can learn from [63, 87, 89]. The issue, then, is how well the model can predict the expensive experiments since it lacks a large amount of training data to try and model a pattern from. It would almost become an extrapolation problem at that point due to this – which is much more difficult than interpolation [14, 87].

With transfer learning, the goal is to combine the respective knowledge of the expensive and cheap datasets to create a final model that does well with "expensive" predictions. Consider a concept referred to as *PerfNet Architecture* [89] which has three neural networks (as described in section 2.4): source, target, and final. The first step is to train the *source* network on the "cheap" dataset so that it nearly perfectly fits (overfits) the data. Then, the *target* network (which is untrained at this point) is appended onto the *source* network such that the output(s) of the *source* network act as inputs to the *target* network. In addition to this, the parameters of the *source* network (ie. biases and weights) are frozen so that they can't be updated any further [75, 89]. Then, the "expensive" dataset is input to this resulting network and trained until the outputs reach an accuracy threshold set by the user. The result is the *final* network that can then be used for predictions. In the case of the ICF experiments [75], they only used 20 "expensive" instances to train the network with a handful left to test the model.

The resulting model preserves the knowledge gained from the "cheap" dataset while appending the small amount of knowledge gained from the "expensive" dataset. This is better than simply making a surrogate model for the entire dataset as that will introduce uncertainties when it comes to the "expensive" predictions that are more desirable [75, 87, 89]. Similarly, if a neural network was simply trained on the entire dataset, it would lose knowledge gained from the "cheap" dataset as the "expensive" dataset would introduce noise

that the network would try to accommodate for [87]. Experiments performed in [89] showed the latter to be true by testing it on a particle physics simulation known as *Kripke* [74]. They found that, for this problem, random forests (as described in section 2.4) performed the best out of the models they tested but that a network created with transfer learning performed significantly better than that [74]. This technique will be used in Chapter 5.

In the chapters to follow, articles that use machine learning to resolve social and health issues that are either published, in proceedings, or in progress, will be presented. The next two chapters (3 and 4) will describe a deep-learning model for agent-based populations, with a specific focus on homeless individuals. The motivation for this work was to provide policy makers with a deep learning solution to test different approaches at assisting this population.

Chapter 3

Simulating the Evolution of Homeless Populations in Canada Using MDQL and MNFQ Algorithms

All of this chapter was published in the following peer-reviewed journal article [35]:
• Fisher, A., Mago, V., & Latimer, E. (2020). Simulating the Evolution of Homeless Populations in Canada Using Modified Deep Q-Learning (MDQL) and Modified Neural Fitted Q-Iteration (MNFQ) Algorithms. IEEE Access, 8, 92954-92968.

This publication captures my contribution to a larger research initiative that applies artificial intelligence techniques to modeling/predicting variables in a population of homeless people assisting a Canadian city in distributing support. I modified pre-existing artificial intelligence techniques to create a novel approach in simulating homeless individuals as the core of a platform used by policy makers. The main changes to these algorithms were as follows: removing the "action" step in q-learning, modifying q-learning's concept of "rewards" to utilize a training dataset, and updating the q-values in a way that reflects a transition probability matrix. I took the lead on developing this publication with the support and guidance of my thesis supervisors. We elected to publish in IEEE Access because of its peer-review process and open-access policy which was beneficial for my first publication at the university.

3.1 Introduction

The Problem of Homelessness

Homelessness is a source of growing concern across Canada as well as in most developed countries [123], with numbers increasing in most Canadian cities [19, 23, 78, 124, 125, 129] and internationally [27, 85, 31]. It is estimated that on any given night, about 567,715 people are homeless in the United States [27], and 35,000 in Canada [40]. Homelessness is associated with worse physical and mental health [33, 61], increased mortality [109], greater criminal behavior and victimization [110], and high health and criminal-justice-related costs [79].

The causes of the rise in homelessness are not completely understood but certainly include rising income inequality [76] together with rising rents in many areas [48]- leading to a growing shortage of affordable housing. At an individual level, many factors predispose a person towards homelessness, including low educational attainment, joblessness or low income, poverty, mental illness, and substance abuse [100, 130]. In between these macro- and micro-level factors are institutional arrangements such as lack of adequate supports for people who were previously homeless or who are at risk of homelessness who leave the youth protection system, prisons, and hospitals [100]. These factors interact with each other in a complex manner [100].

In recognition of the complexity of the phenomenon, numerous policies and programs, operating at different levels, have been put in place to address homelessness. These have included increasing the availability of affordable housing (whether through building new affordable housing units or providing low-income individuals with rent supplements); helping individuals who have become homeless regain permanent housing, through a variety of more or less intensive and short- or long-term supports, notably Housing First, which offers in-

dividuals a combination of a rent subsidy and the long-term support of a mobile clinical team [64, 127]; and an array of primary, secondary and tertiary prevention measures [41]. The complexity of the phenomenon, together with the wide range of possible remedial and preventive measures, means that addressing homelessness effectively is challenging, and experts disagree on many aspects of the policies that should be pursued in a given city or geographical area.

Computer Simulation Modeling to Help Address Homelessness

Computer simulation modeling offers a possible decision support tool to address homelessness. This approach has often been used to try to gain deeper insight into complex problems of many kinds [120]. Limited attempts in this direction have been made until now with regards to homelessness, however. These attempts can be classified, to date, into 4 groups: (a) economic models calibrated to individual cities [88]; (b) entirely mathematical models that do not incorporate any city- or area-specific empirical data; (c) mathematical models based on a survey of empirical results found in the literature [86]; (d) statistical models that relate the number of homeless individuals in an area to a number of other area-specific variables. Among these, only the first has the potential to simulate the effects of alternative specific policies on the number and composition of homeless individuals in an area. The one study of this type that the authors have identified is, however, entirely focused on the housing market and the effects of alternative housing subsidy mechanisms; It does not take into account programs to help homeless individuals access housing, such as Housing First; nor does it distinguish among the different states that homeless people can find themselves in, such who experienced a one-time, brief episode in a homeless shelter, or someone who has been alternating for years between sleeping in street locations and in shelters. This was

the main objective of this project.

Research Challenges

Constructing such a fine-grained model poses two main challenges: how to structure the model, and where to get the data to calibrate it. The authors chose to structure it as a Markov model, with a cycle length of one week, in which individuals can be located in any one of 8 states: street location, shelter, transitional housing, substance abuse treatment center, hospital, prison, not homeless (but previously so), or deceased. Individuals transition from one state to the next on the basis of a set of transition probabilities. The capacity of shelters and transitional housing in a city, by age group (25 and under or over 25) and gender, is input into the system, and occupancy of these resources cannot exceed 100%. (The other types of institutions mostly serve non-homeless individuals and their capacities can thus, for this chapter's purposes, be considered unconstrained.)

To derive the transition probabilities, this work used data from the Montreal site of a large Canadian study, the At Home/Chez Soi trial. This site followed 463 initially homeless individuals up to two years at a time, reconstructing their day-by-day housing trajectories. Thus it provides data at a resolution sufficient to enable simulation of each individual in a cohort separately. This is necessary given the work's desire to incorporate shelter and transitional housing capacity constraints. Recruitment for this study took place, however, between 2009 and 2011, and the sample was not designed to be representative of the homeless population as a whole. Furthermore, the service system in Montreal has evolved since the early 2010s, notably with the progressive addition of Housing First programs with a combined capacity of several hundred.

In order to calibrate the model, the authors had access to data from Montréal's March

24 2015 and April 24 2018 point-in-time homelessness counts, which provide not only the number of homeless individuals at those dates, but detailed data on their locations and demographic characteristics. The authors wanted, therefore, to base the work on the transition probabilities to reproduce the evolution of the number and composition of the homeless population from the first count to the second. A method is needed for adjusting the transition probabilities so that they fit the point-in-time data. As some individuals left the system (due to death or exiting homelessness), others also enter it, becoming homeless for the first time and, in some cases, remaining homeless for the long term. This also needs to be represented in the model. To this end, some data were available from another survey of homeless individuals conducted in Montreal five months after the first point-in-time count. These data provided information, for individuals who were homeless on August 24, 2015, on whether they were homeless on March 24, 2015, and if so, in what type of location. This work's basic approach was to use two modified q-learning algorithms to adjust the transition probabilities so as to be able to reproduce, as closely as possible, the 2018 count data starting with those from 2015. These counts are described and explored further in the next chapter.

Summary of the Algorithms and Contributions

The proposed simulation model works with two, modified deep-learning algorithms; the originals being deep q-learning[28] and neural fitted q-iteration[107]. Originally, the algorithm would start in an initial state, perform an action, and observe the new state it transitioned to [28], to determine a reward [62]. Instead, with the proposed simulation, the action performed is simply determining which *new state* to transition to; the algorithm is picking the new state based on the transition probability matrix. The reward for the algorithm is determined by what the current populations in the simulation are and what the final populations

should be (from the training data) after the simulation has ended.

For neural fitted q-iteration, the original algorithm worked with deep q-learning by performing an offline update to determine the best action to perform in the algorithm's current state. It is offline in a sense that all *previous* changes are considered when doing it [107]. The main modification the authors made here was that, instead of determining the best action, the algorithm calculates an *offset* to modify the new q-values by, based on previous changes. This offset helps lower or increase the q-values, which are later interpreted as the transition probability matrices for each state transition pair. In a sense, the result is the best action that the algorithm determines to ensure realistic transitions between states.

To summarize, this work made the following modifications to both the deep q-learning and neural fitted q-iteration algorithms:

- Removed the "action" step
- Modified the purpose of the reward term
- Modified the neural fitted q-iteration to make it more versatile for other applications

Organization of the Chapter

In the next section, this chapter examines related work to the proposed model. The authors discuss recent papers on homelessness issues, modelling approaches, and machine learning in social sciences to show how this family of algorithms has been beneficial in other implementations. In the subsequent section, the authors look at the methodology of the proposed approach and describe each modified algorithm in detail. Lastly, the authors provide a discussion of the results and propose future work to improve the model. This section will also include results when the model was applied to Montreal's 2015 and 2018 homelessness count

data, using transition probabilities derived from the Montreal site of the At Home/Chez Soi project [3, 119].

3.2 Related Work

Previous Efforts to Model the Phenomenon of Homelessness

As mentioned earlier, previous efforts to model the phenomenon of homelessness can be classified into 4 groups: (a) economic models calibrated to individual cities [88]; (b) entirely mathematical models that do not incorporate any city- or area-specific empirical data; (c) mathematical models based on a survey of empirical results found in the literature; (d) statistical models that relate the number of homeless individuals in an area to a number of other area-specific variables.

Authors of [88] provide, to the authors' knowledge, the only example of the first group of efforts. This chapter describes it in more detail as it is the only one that in some ways approximates what this work is trying to do. The authors used a general equilibrium model of the housing market to examine policies to reduce homelessness, calibrated for four California cities. Theirs is a model of the housing market, in which "dwelling units filter through a quality hierarchy... and in which households of various income levels choose among these discrete types." Households may choose to opt out of the housing market and thus become homeless. They are more likely to do so if available housing is unaffordable to them. Thus, increases in homelessness are driven by changes in rents. They concluded that "a very large fraction of homelessness can be eliminated through increased reliance upon well-known housing subsidy policies", in particular, rent subsidies [88].

Authors of [38, 96] provide examples of the second type of model. As these models do

not incorporate any empirical data, they are of limited value as a decision-support tool for a particular city. Alone to the authors' knowledge in the third group, [86] applied a well-established technique called fuzzy cognitive maps to analyze the impact of social factors on homelessness. Their macro-level model, which was calibrated using information extracted from the literature, was able to reasonably represent reality for a range of scenarios. The direction and strengths of the relationships between concepts included in the map approximated their action in reality. Education emerged as having the greatest force in the model. Again, however, the model remains general, and not useful as a decision-support tool for a particular city [86].

Finally, the fourth group is comprised, again, of a single effort. [58] led the development of the "Homelessness analytics initiative"[1]. This web site compiles a large amount of data on homeless counts in the many US areas where these are now regularly carried out, and social (e.g., crime) and health (e.g., county-level life expectancy) indicators as well as other contextual factors (e.g., fair market rents). The web site also provides access to a set of forecasting models. These models are based on regression analyses of homeless counts against social indicators and other predictor variables. However, once again, these statistical models, being based on commonly available social and economic indicators, are of limited usefulness in estimating the effects and costs of alternative policies in a particular city or area.

Mathematical Modelling

A mathematical modeling presented by Zhang, T. *et al* at the Winter Simulation Conference in 2018 where the main issue they addressed was that previous attempts at automating batching in job shops were falling short of their goals in real-life environments. The batching,

[1] `http://homelessnessanalytics.org/`

here, is done when several jobs are processed *simultaneously* [139]. The approach was due to the fact that the typical job was too complex in nature and not fit for a stochastic environment; that is, the jobs lacked a linear pattern that could be easily modelled. Practitioners also preferred real-time batching where the scheduling was considered as more of a decision-making problem. To solve this problem, this chapter introduces a sequential decision-making process using Markov decision processes.

Another mathematical model by Batata, O. *et al* focused on optimizing care resources by predicting the burnout in a caregiver to admit them to respite services before hospitalization is needed [6]. The main issue was that previous attempts fell short when it came to actually predict the number of patients needing to be admitted. Respite care is a new service that aims to help decrease the burnout risk in caregivers. When this is not taken care of in time, the caretaker eventually needs hospitalization, which is costly. This chapter talks about the need to be able to predict burnout in a caregiver and admit them to respite services before hospitalization is needed.

Authors of [6] attempted to use the addition of machine learning as well as Markov chain transition matrices to create a dynamic burnout model with two states for the caregivers: emergency and normal. By having the ability to see what a caregiver's next state will be, the system could efficiently decide whether or not they should be admitted into respite services. From their experiments, Batata et al's model performed best using a *neural network*; the objective was to minimize the number of hospitalized individuals. One major shortcoming of this work, however, was that the Markov chain was strictly built using only burnout data without considering attributes specific to the caregiver and their patient.

Machine Learning and Simulation

Authors of [62] introduced a new approach using machine learning and simulation to optimize the dose calculation for radiotherapy. This approach involved two steps: an agent-based simulation of vascular tumor growth and a q-learning algorithm to optimize the radiation dosages. The optimal outcome when combining these two steps is to achieve a cure for tumor with minimal side effects. The researchers noted that many studies had been done in the area of radiotherapy simulation, and optimization but not in optimizing radiotherapy based on simulation. Due to a lack of real data, the agent-based simulations noted before were ran to help the generation of synthetic data needed for the optimization. The outputs allow interfering in the simulations to examine different scenarios. Since this area has not been researched much, it is hard to note any shortcomings. Two points of consideration: there was no real-life experiment, and this approach uses a process they called inverse planning. This process is aided by a computer to test (simulate) different treatment plans before any physical experimentation is done [62].

An important theme amongst recent papers in this area involved q-learning. More precisely, neural fitted q iteration (NFQ) [107]. The main issue that this algorithm addressed was an issue that arose with multi-layer perceptrons when training them. During the training process, when modifying a parameter change in one area of the network, the algorithm has the potential to influence other values later on in the network. The approach inadvertently destroys the effort done so far in other regions and leads to long learning times or, worst case, not learning at all. In order to solve this limitation, when updating the q-value functions, the algorithm offers previous knowledge explicitly as well by storing all previous experiences in terms of state-action transitions in memory.

To implement their approach, the algorithm uses what's called an off-line update rule.

In other words, the algorithm considers an entire set of transition experiences- contrary to traditional q-learning, which uses an on-line update rule. This strategy gives the advantage of applying advanced supervised learning methods to the network. For example, [107] uses Rprop, which is very efficient and very insensitive to the learning parameters. NFQ falls under the fitted q-iteration family of algorithms; it is a memory-based method used to train q-value functions based on multi-layer perceptrons. By exploiting generalization, the algorithm is able to achieve a high level of data efficient learning. This chapter provides three real-life scenarios that are quite diverse in specifications. This shows that the algorithm is applicable to a wide variety of tasks and can work well with real-life scenarios.

3.3 Methodology

Data Source

The work in this chapter uses data from the At Home/Chez Soi project [3, 119] to train the proposed algorithm. Researchers gathered the data of individuals who entered and exited different states (street, shelter, etc.) at different points in time over a year. Interviewers tracked these movements using retrospective questionnaires administered every 3 months. This method of data collection makes it possible to generate transition matrices on a week-to-week basis. Since each person will be simulated individually, this work can accommodate the exact information of data available to us. In the chapter to follow, this dataset will be further described as the intricacies present were not considered for this work at the time.

Data Pre-processing

Transition probabilities were estimated from data collected at the Montreal site of the At Home/Chez Soi project [81]. This was a large (n=2,148) randomized controlled trial of Housing First for homeless individuals with mental illness, compared to usual services, conducted in five Canadian cities: Vancouver, Winnipeg, Toronto, Montreal, and Moncton. Housing First offers individuals experiencing homelessness immediate access to a choice of subsidized, rental market apartments, together with the support of a mobile team of mental health and other professionals [119]. Study participants were all people with mental illness, who had been homeless for varying lengths of time and who expressed an interest in being housed. They were recruited between October 9, 2009 and May 31, 2011. They were interviewed at 3-month intervals for up to 2 years. A questionnaire was used to reconstruct places they had been each night since the previous interview [51].

Machine Learning Based Algorithms

At a basic level, this approach can model the simulation using a Markov decision process with a transitional probability matrix. This matrix is calculated by looking at the number of individuals who go from one state to another, then calculating the probability that this will occur based on the total population in that initial state. An example set of transition probabilities can be seen in Table 3.1. Using such a model would essentially be a simplified, mathematical representation of reinforcement learning [139]- a common approach when training machine learning models.

By using a mathematical model, such an approach would need to monitor the outputs and see if any changes are needed for the model. This process can become cumbersome when working with large datasets, such as a population of people, as the values would, initially,

need to be updated constantly to produce accurate results. To address this problem, the authors create a machine learning model that, although requiring a small amount of reinforcement learning, will mostly be unsupervised when training on real-life data. The small amount of reinforcement learning here is from the value of an *offset* and if it is minimizing from previous iterations. An example visualization of these probabilities and how they may change from one time period to the next can be seen in Figure 3.1, which shows the values of a basic set of transition probabilities from one time period to the next. The arrows in this diagram represent the transition from one state to another, with their probability noted as the p value. The authors demonstrate the model on data from the At Home/Chez Soi project [3, 119] to show how it learns and modifies the transition probability matrices over time.

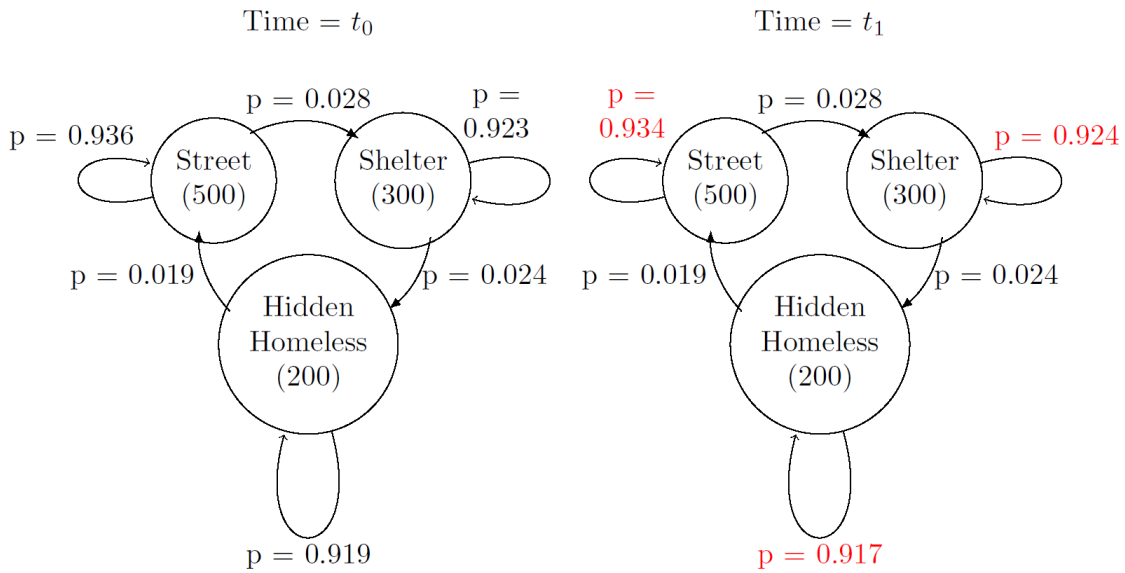


Figure 3.1: A simplified example set of transition probabilities between states and how they may change in a machine learning model. Note that this is not showing all possible transitions [36].

Furthermore, one shouldn't assume that a single transitional probability matrix will

	Street	Shelter	Hidden Homeless	Not Homeless	Transitional Housing	Death
Street	0.936	0.028	0.030	0.006	0.005	0.000
Shelter	0.019	0.923	0.024	0.017	0.017	0.000
Hidden Homeless	0.019	0.029	0.919	0.022	0.011	0.000
Not Homeless	0.006	0.009	0.009	0.969	0.004	0.003
Transitional Housing	0.003	0.008	0.006	0.009	0.974	0.000

Table 3.1: Another example of a simplified set of transition probabilities [36] that is represented in a tabular form.

work when simulating on a higher resolution scale such as week-to-week. To address this assumption, the model will, instead, generate a probability matrix for each week of the year. Although the goal of this model when training is to accurately predict the known final populations, the authors are also wanting to use it to predict realistic results for future populations where the end population is unknown.

Homelessness Simulation

Over time, homeless individuals can transition between many different states of homelessness. For the proposed simulation, these states include street, shelter, hidden homeless, not homeless (but previously were), transitional housing, hospital, rehabilitation (drug/alcohol), and prison.[2] The prediction that an individual in an initial state s_i will transition to a new state s_n can be defined as a conditional probability $P(s_n|s_i)$ as given in Equation 3.1:

$$P(s_n|s_i) = \frac{N(s_i, s_n)}{N(s_i)} \tag{3.1}$$

where $N(s_i, s_n)$ is the number of people transitioning from s_i to s_n and $N(s_i)$ is the

[2]Adapted from the At Home/Chez Soi project [3, 119]

number of individuals in s_i. An individual can transition from their initial state to their initial state (no change) or from one state to another state (by the end of week), for instance, street to shelter. Therefore, for the states defined before, one would have 72 probabilities (a matrix of 8 initial states and 9 new states) for each state-to-state transition–excluding death to another state. However, from just analyzing the data alone, the proposed approach would only take into account the *change in populations*. Other outside factors can also affect this probability. Consider the following graphs where the authors analyzed the cumulative population of six males (Figure 3.2) and seven female (Figure 3.3) shelter populations in Montréal,[3] respectively, over the course of a year as well as the temperature (Figure 3.4)[4] over that time period. Here, one can clearly see an impact on the shelter population of temperature. This is an example of how simply calculating the transition probability matrices for two points in time will not suffice to create an accurate simulation. This chapter proposes a model where the transition matrix probabilities are *dynamically updated* based on the current state of the system and what the end result should be when training.

MDQ-Learning

The proposed approach will be interpreting the q-values from q-learning (Chapter 2.4) as transition probabilities for each state-to-state transition. One of the algorithm's main difference from deep q-learning is that it only uses *one neural network*; loss will be calculated by looking at the previous output to see how it is converging. Another difference is how this work will process the state transitions. Unlike deep q-learning, the data being used does not necessarily have an *action* that leads to a new state; it is a direct transition.

In order to simulate each member of the population accurately, the authors introduce a

[3]Data provided by the city of Montréal

[4]Data retrieved from https://montreal.weatherstats.ca/metrics/temperature.html

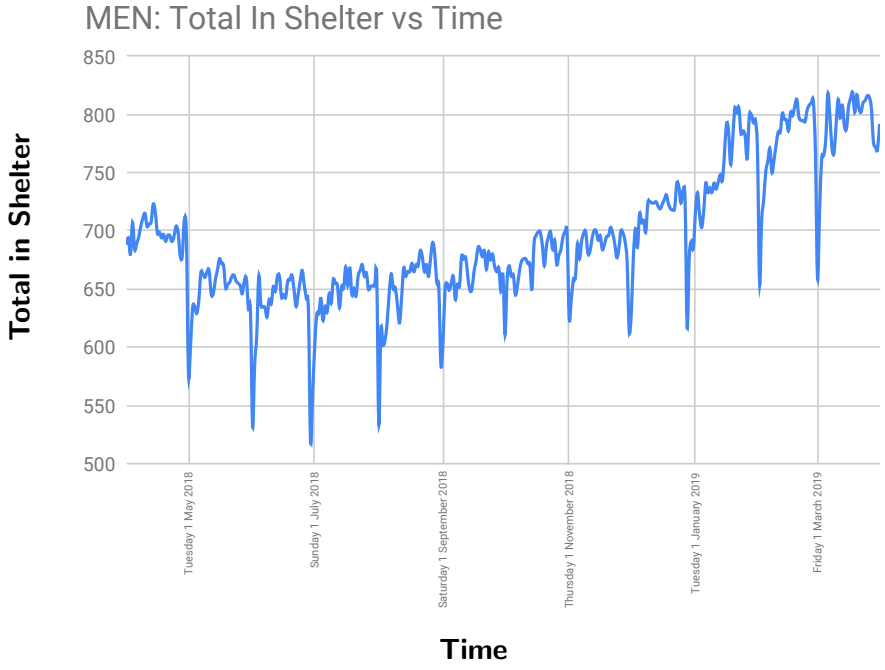


Figure 3.2: The change in Montréal men's shelters' population over time.

sub-epoch that occurs on every *epoch* in the algorithm. This sub-epoch will run each member, through the network, with their state and action, producing an ideal q-value for them *individually*. To get the reward for this new q-value, consider the following in Equation 3.2:

$$R_s(s_i, s_n) = 1 - \frac{Q_e(s_i, s_n)}{Q(s_i, s_n) + Q_e(s_i, s_n)} \tag{3.2}$$

where $Q_e(s_i, s_n)$ is the calculated q-value at the *current* epoch and $Q(s_i, s_n)$ is the q-value for that state-action pair as of the *previous* epoch. The actual q-values– $Q(s_i, s_n)$– are derived from the actual data as per Equation 3.1.

In deep q-learning, the goal is to maximize this reward. Since the goal is, instead, to produce the best *q-value* for this transition, one can see an obvious problem with this formula. If the network needs to lower the value of a probability, it would be difficult with this reward

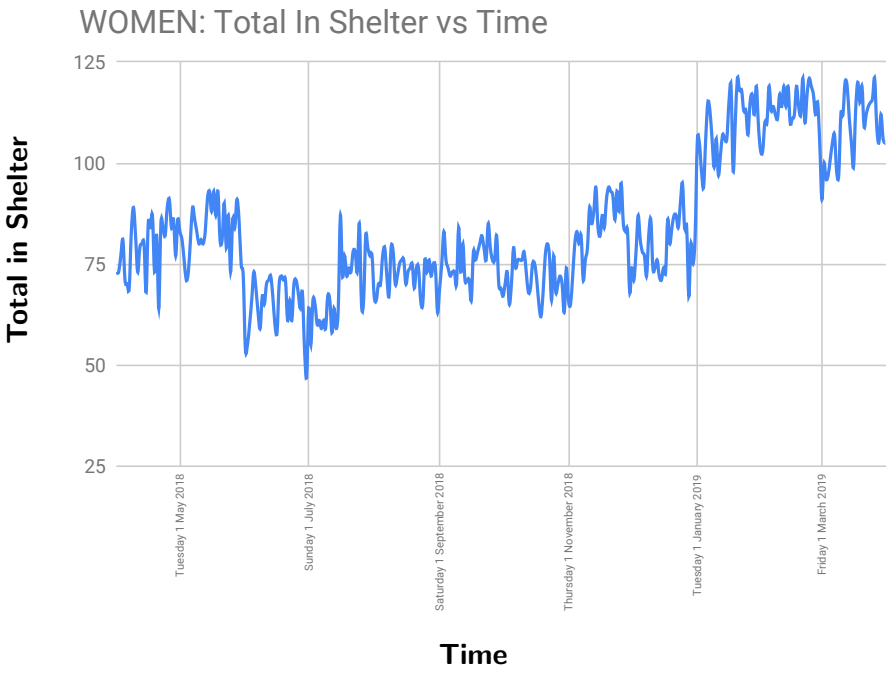


Figure 3.3: The change in Montréal women's shelters' population over time.

formula as it only produces positive values. This topic is addressed in the MNFQ-Learning section.

The authors introduce a simple approach in determining the new states in a population commonly known as a *roulette wheel*. Consider the following in Equation 3.3:

$$Q(s_i, s_n) > \sum_{j=1}^{x} Q(s_i, s_{n_j}) \tag{3.3}$$

where x is the total number of new states and i is the current state the individual is in. This definition assumes that the q-values have been normalized between 0 and 1. To select the new state, one can simply generate a random number between 0 and 1 then see where it falls in the range (denoted by the $>$ symbol).

For example, if you had three probabilities 0.25, 0.35, and 0.4, the wheel would look like

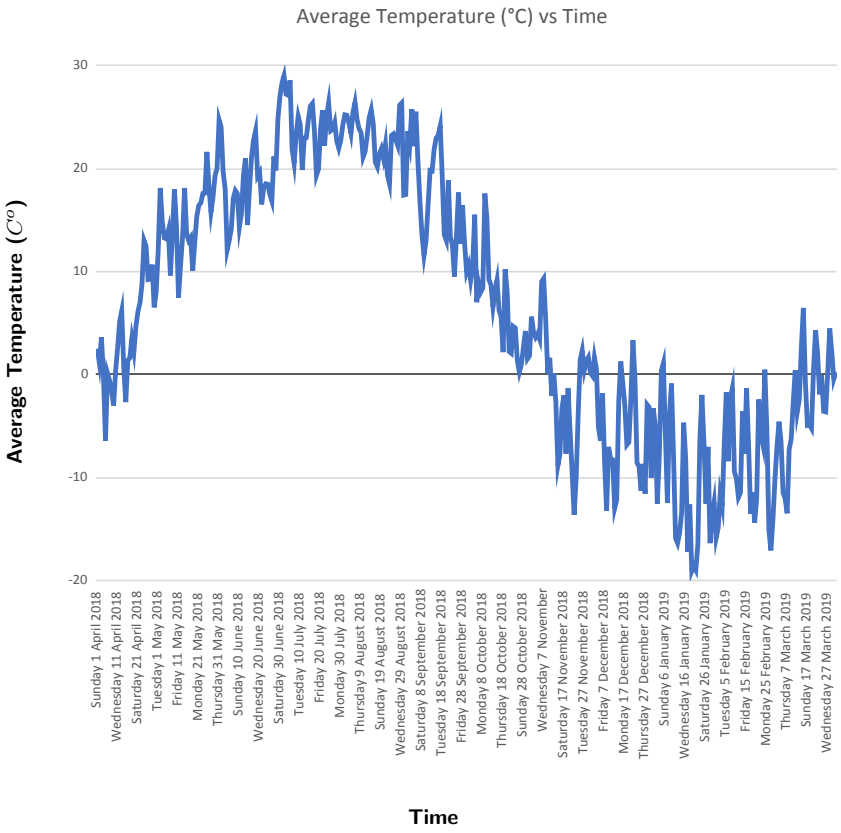


Figure 3.4: Montréal's temperature over time from mid 2018 to mid 2019.

the following in Equation 3.4:

$$Wheel = 0.25 > 0.60 > 1.00 \tag{3.4}$$

Then, the model would generate a random number, for example 0.55. Next, the model would identify if it falls between 0 and 0.25. If it does not, it would go to the next range which is 0.25 and 0.60; which would be a match. So, the selected action would be action 2.

Once all of the sub-epochs have ran, the approach can calculate the new q-values for the algorithm. The reason for this process is to ensure that all datapoints from every individual is accounted for when training as an **epoch** covers the entirety of the dataset while a **sub-**

epoch covers each dataset for each individual. Consider the following in Equation 3.5:

$$Q'(s_i, s_n) = Q(s_i, s_n) + [1 - \frac{\sum_{j=1}^{N} R_j(s_i, s_n)}{N}]\eta \tag{3.5}$$

where $Q'(s_i, s_n)$ is the new q-value, $Q(s_i, s_n)$ is the previous q-value; N is the size of the population; R_j is the reward for each member of the population in that transitioned from state s_i to state s_n, and η is the learning rate. This equation essentially takes the average of the rewards from the maximum reward, multiplied by the learning rate, as the value to update the q-value by. After all of the updates have occurred, the approach will then normalize the results to ensure they still fall in between 0 and 1.

The last consideration is how the network is trained. Consider the following in Equation 3.6 for the error:

$$E_{out} = \frac{\sum_{j=1}^{x}[Q'_j(s_i, s_n) - \frac{\sum_{k=1}^{N} Q_{e_k}(s_i, s_n)}{N}]}{x} \tag{3.6}$$

where x is the number of q-values in the table and Q_{e_k} is the outputted q-value for each member of the population that transitioned from state s_i to state s_n. This aspect is calculating the average difference between each new q-value and the average q-value that the population outputted. This error will then be applied to the output neuron of the network so that back-propagation can be performed.

In deep q-learning, the goal is for the prediction network and target network to converge. Since the implementation only uses one network, this work will simply look at how the q-values are converging from one epoch to the next. Consider the following definition in Equation 3.7:

$$Loss = \frac{\sum_{j=1}^{x}[\frac{\sum_{k=1}^{y}[Q'(s_{i_j}, a_{n_k}) - Q(s_{i_j}, a_{n_k})]^2}{y}]}{x} \tag{3.7}$$

Here, it is getting the average of the average of the differences between the new q-values and the old q-values squared where x is the number of states and y is the number of actions.

To input into the states to the network, the approach will simply create an array of them and assign their index in the array as their value to the network. Therefore, the network takes two inputs: the index of the state of the person and the index of the state to which they are transitioning. The output is an optimal q-value based on the current state of the model for the current member of the population. The proposed model has a network with four hidden layers having 4, 8, 16, and 24 nodes respectively. The authors do give the option of modifying the size and count of the hidden layers here, but this is the authors' suggestion from testing the model. For example, if one had the states street, shelter, and hospital, street would be recognized as state 0, shelter as state 1, and hospital as state 2 to this neural network. As for an initial transition matrix, the authors suggest manually analyzing the input data first and calculating a matrix to speed up the algorithm's training. These probabilities can be calculated using Equation 3.1 in the homelessness simulation section.

MNFQ-Learning

In order to determine an accurate q-value (or transition probability for this work's purposes) for the next epoch, this algorithm will calculate an offset to add onto the calculated q-value from MDQL. Consider the following in Equation 3.8:

$$Q'(s_i, s_n) = Q(s_i, s_n) + [1 - \frac{\sum_{j=1}^{N} R_j(s_i, s_n)}{N}]\eta + \delta Q(s_i, s_n)\eta \tag{3.8}$$

This is the same equation as the new q-value in the MDQL algorithm but with the offset, $\delta Q(s, a)$, that MNFQL will calculate. The probability (or q-value) is accurate based on whether or not it is *realistic* for the training data.

The way this offset will be calculated for each $Q(s_i, s_n)$ is by considering all of the previous q-values outputted by the MDQL algorithm for that state-action pair, the current population in the new state, the known end population for new state, and the weeks left in the simulation. To increase accuracy with the q-values, the approach will use a separate neural network for each state-new state pair. Each network is individually trained with the data for its respective state-new state pair, before training the primary MDQL algorithm as a result. For example, if this problem was applied to the probability set in Table 3.1, the result would be $5 \times 6 = 30$ networks.

Each network will take the following inputs: each q-value previously calculated for that pair individually, the population in the new state, weeks left in the simulation, and the current q-value. The output of the network will be the end population for the new state. The q-value that gives the lowest error will be defined as α, and the output error from that will be used for back propagation in the network. This error, E, is simply the difference between what the network outputs and the known end population. For the offset, consider the following in Equation 3.9:

$$\delta Q(s, a) = \frac{\alpha}{\sum_{j=1}^{x} Q(s_i, s_n)} \left(\frac{-E}{N_{s_n}} \right) \tag{3.9}$$

where N_{s_n} is the known end population size for the new state. Since the network is taking the total population in the new state, it will need to times by the best q-value divided by the sum of all states with this new state (the summation would exclude the state-new state pair that α is a part of and add α on instead) to get an estimated percentage that this state transition will contribute to the final population. This equation will then give us the relative, percent difference in the end population, multiplied by this.

For the rewards in MDQL, the authors propose the following modification to the formula

as shown in Equation 3.10:

$$R_s(s_i, s_n) = \sigma Q(s_i, s_n) \tag{3.10}$$

where $\sigma Q(s_i, s_n)$ is the *immediate* value from $\delta Q(s, a)$. This uses the same formula for $\delta Q(s, a)$, but α (previously the q-value that gives the lowest error) is, instead, the q-value calculated for the current individual. If the error from this is less than 20%, back propagation is performed on the state, new state network.

3.4 An Example

Consider a set of ten (10) individuals that can only be in one of three of the following *homelessness* states: shelter, street, and hospital. These individuals are divided up as shown in Figure 3.5.

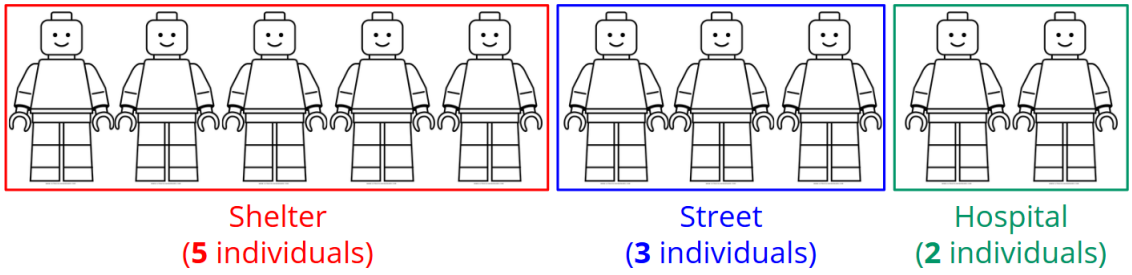


Figure 3.5: An example homeless population that consists of 10 individuals across 3 states.

For each of the states, the algorithm "sees" the following: shelter as 0, street as 1, and hospital as 2. Therefore, for the population in figure 3.5, the algorithm would "see" it as an array as follows: {0, 0, 0, 0, 0, 1, 1, 1, 2, 2}.

Initialization

Consider the population in the previous section and, for the sake of simplicity, say that a year has 4 weeks (or quarters). This example will have two (2) years of data as shown in Figure 3.6.

Year 1

	Shelter	Street	Hospital
Week 1	5	3	2
Week 2	4	4	2
Week 3	4	2	4
Week 4	3	5	2

Year 2

	Shelter	Street	Hospital
Week 1	6	3	1
Week 2	5	4	1
Week 3	5	1	4
Week 4	2	8	0

Figure 3.6: An example dataset of a homeless population over time.

The algorithm would "see" the data in Figure 3.6 as follows:

YearOne = {0, 0, 0, 0, 0, 1, 1, 1, 2, 2}, {0, 0, 0, 0, 1, 1, 1, 1, 2, 2}, {0, 0, 0, 0, 1, 1, 2, 2, 2, 2}, {0, 0, 0, 1, 1, 1, 1, 1, 2, 2}

YearTwo = {0, 0, 0, 0, 0, 0, 1, 1, 1, 2}, {0, 0, 0, 0, 0, 1, 1, 1, 1, 2}, {0, 0, 0, 0, 0, 1, 2, 2, 2, 2}, {0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 2}

Since the algorithm is presented with three (3) states (shelter, street, and hospital) and four (4) quarters in a year, it will initialize the transitional probabilities for each quarter (4 sets in total) as shown in Figure 3.7 (these were randomly generated with the highest probability given to stay in the same state).

The next step of the initialization is to create one (1) neural network for MDQ-Learning

Current State \ New State	Shelter	Street	Hospital
Shelter	0.72	0.22	0.06
Street	0.32	0.58	0.10
Hospital	0.27	0.22	0.51

= 9 probabilities = P

Figure 3.7: An example transition probability matrix for a homeless population across 3 states.

(used to simulate) and P (number of probabilities) individual neural networks for MNFQ-Learning (used to train the model).

Training: Initial Loop

The algorithm starts by cycling through the data for each week of each year. Let's follow the first week of year one:

Input population = {0, 0, 0, 0, 0, 1, 1, 1, 2, 2}

This population is passed to the MDQ-Learning algorithm to simulate each person *individually*. Let's follow the first person who is in the shelter state; consider an example transitional probability matrix for this week of the year (week 1) in Figure 3.8.

Since the individual being followed is in the shelter state, the approach is only concerned with the probabilities that they will transition to other states *from* their initial state. Therefore, transitioning to the shelter has a 72% probability of occurring, street a 22% probability, and hospital a 6% probability. With that in mind, the new state for this individual is deter-

Current State \ New State	Shelter	Street	Hospital
Shelter	0.72	0.22	0.06
Street	0.32	0.58	0.10
Hospital	0.27	0.22	0.51

Figure 3.8: An example of what the first week's transition probability matrix looks like in this section's example.

mined using a weighted roulette wheel based on these probabilities as shown in Figure 3.9.

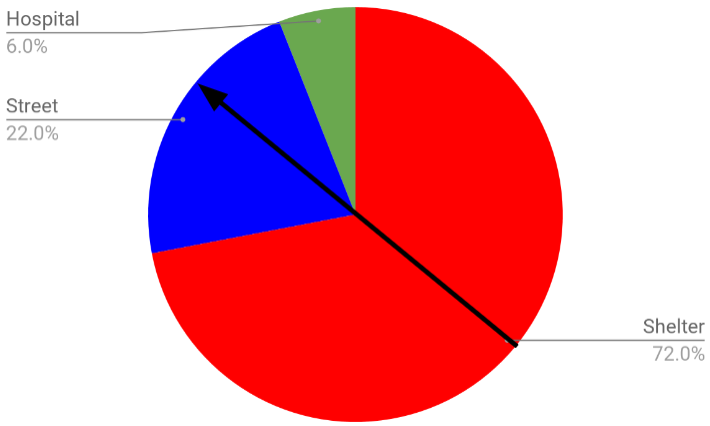


Figure 3.9: A sample, weighted roulette wheel that follows the transition probability matrix for this section's example.

The wheel is "spun" (a random number is generated) to determine which state to transition this individual to. The new state is recorded (for the individual being followed, street), and input into the MDQ neural network with the previous state (for the individual being followed, shelter) to get a *recommended* transition probability for the individual. This output will be defined as $Q_e(s_i, s_n)$, where s_i is the initial state, s_n is the new state, and Q_e is

the output. Consider Figure 3.10 which shows a sample MDQ network.

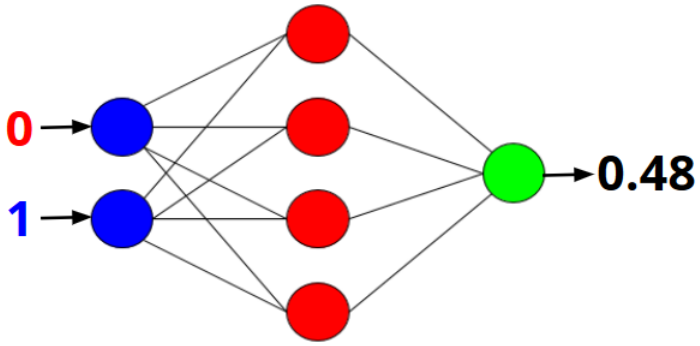


Figure 3.10: An example MDQ network that takes the previous and current state of an individual as an input and outputs an optimal transition probability.

Next, with the MNFQ neural network for the current individual's transition (in this case, the shelter → street network), it takes the following inputs:

(1) The transition probability calculated for the current individual, normalized with the current value (from Figure 3.10, 0.48)

(2) The current population in this new state (in this case, after the transition, there's now 4)

(3) The weeks/epochs left to the next population (in this case, the next set of available data is 1 quarter away)

The goal is to see the output's error from the end population (start of next week) in this new state (for street, 4). This error will be referred to as E.

The error is used to determine an immediate *reward* for choosing this transition for this individual with the previously described formula:

$$\delta Q(s,a) = \frac{\alpha}{\sum_{i=1}^{x} Q(s_i, s_n)} \left(\frac{-E}{N_{s_n}} \right) \tag{3.11}$$

Where α is the calculated transition probability (from the previous output, 0.48), N_{s_n} is the end population (for street, 4), and the summation is of all the transition probabilities for each state $\rightarrow$ street probability. Since the goal of the network is to output the total population in the new state, it will calculate the current individual's calculated transition probability divided by the sum of all transitional probabilities to this new state (state $\rightarrow$ street) to get an estimated percentage that the current state transition (shelter $\rightarrow$ street) will contribute to the overall, relative percent error of the final population (in this example, street population). The goal of the algorithm is to *minimize* this value.

This "reward" as well as the calculated transition probability is recorded for each individual. If the error from before, E, is less than 20%, the MNFQ neural network (in this case, the shelter $\rightarrow$ street network) is trained by performing back-propagation. Once all individuals have been processed, the algorithm uses each MNFQ network to determine an *offset* to add to each transition probability in order to better fit the data.

Consider the same equation described for the "reward" value previously but this time, however, α is instead the calculated transition probability, from each individual, that gives the *lowest error* for this transition. The rest of the equation variables are identical. This value is then used in the previously described equation:

$$Q'(s_i, s_n) = Q(s_i, s_n) + [1 - \frac{\sum_{j=1}^{N} R_j(s_i, s_n)}{N}]\eta \tag{3.12}$$

where $Q'(s_i, s_n)$ is the new transition probability, $Q(s_i, s_n)$ is the previous transition probability, N is the size of the population in the new state s_n, R_j is the "reward" for each member of the population in that transitioned from state s_i to state s_n, and η is the learning

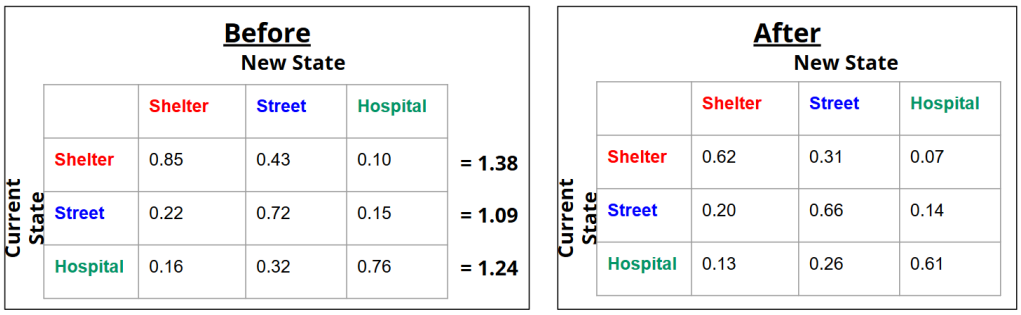
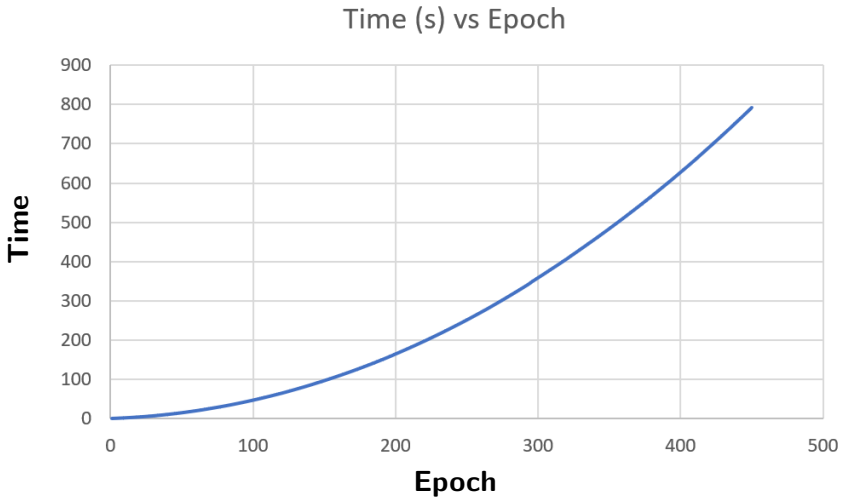


Figure 3.11: A sample transition probability normalization after a training epoch has finished.

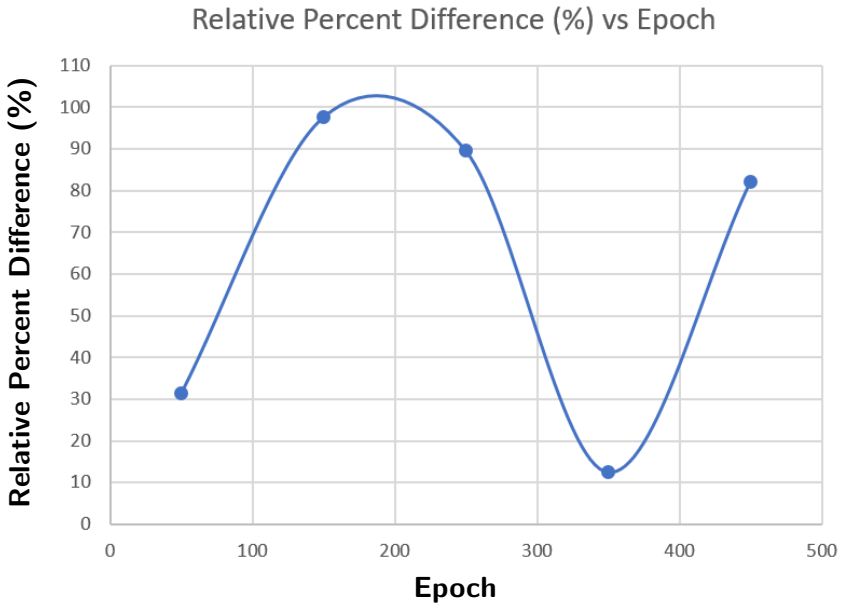
rate. Once all transition probabilities for the current week have been updated accordingly, each row is normalized such that it adds up to one (1). An example of this can be seen in Figure 3.11.

3.5 Performance Evaluation

As discussed in the research challenges, the authors are faced with a lack of data. Using the At Home/Chez Soi project [3, 119] this work created a data set that showed the state for each individual in Montréal at 117 different time-points (or weeks). It should not be assumed that every individual had a state recorded for every time-point; however, this issue did not introduce any complications as the proposed model trains on a week-to-week basis. The authors first trained the model with this data to produce transition probability matrices for each week of the year. This process required us experimenting with the number of epochs and learning rate to determine the best combination for an optimal output. The three learning rates the authors tested with were 0.1, 0.01, and 0.001. After numerous rounds of testing, the lowest relative percent differences from the final population the model achieved were 12.9%, 12.5%, and 66.5% respectively.



(a) A graph showing the overall processing time for training the proposed model after 450 epochs.



(b) A graph showing the relative percent difference of the proposed model after 50, 150, 250, 350, and 450 epochs.

Figure 3.12: Figures showing the performance of MDQL with MNFQ that were trained with a learning rate of 0.01.

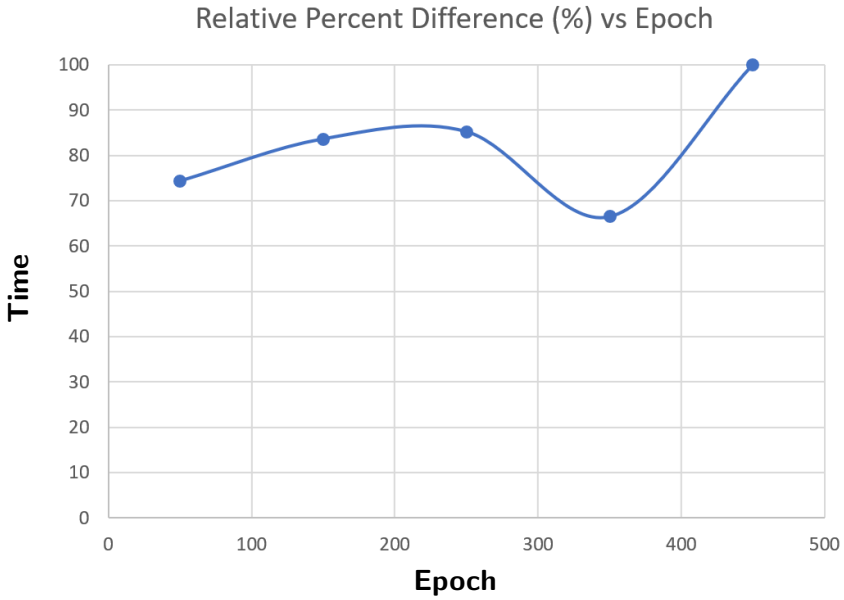
Consider Figure 3.12a that shows the total processing time for the maximum run of 450 epochs at a learning rate of 0.01. It should be noted that for the other two learning rates, the processing time was very similar to this graph. One can see that the time increases exponentially as the epochs are increased. It should be noted that the model is currently using the CPU as the hardware accelerator[5]. Once the model was trained, it produced 52 unique transition probability matrices for each week of the year.

Next, the authors tested the model with 2015 to 2018 homeless state counts from the city of Montréal. Since the model transitions the population on each epoch, the authors consider one epoch to be equal to one week. Therefore, this work ran the populations in 2015 through the trained model for 156 epochs. Consider Figures 3.12b and 3.13 that show the relative percent difference of the final output (i.e., the percent difference from the actual values across all states) for this learning rate, after 50, 150, 250, 350, and 450 epochs.

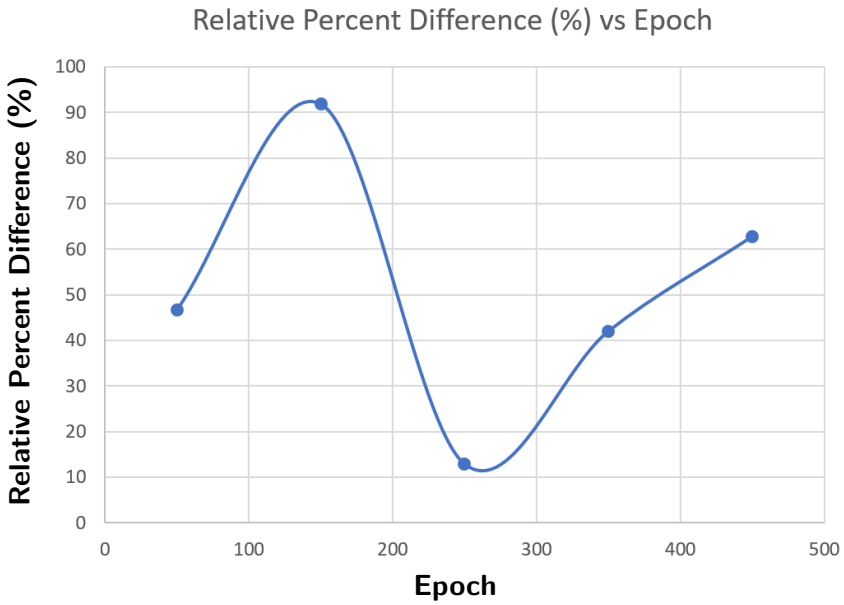
It shows that the relative percent difference of the proposed model followed an almost sinusoidal form. This observation is interesting as lower epochs may be suffice to produce an accurate output. In this instance, at 50 epochs, the model had an relative percent difference of 31.39%. Consider figure 3.14, which shows the best results for each learning rate's best run. At a learning rate of 0.1, one can see that the not homeless and transition housing states were very closely predicted. For the other states, however, there are some differences that need to be improved. The authors discuss implementing a Markov model in the future works section to compensate for this.

To evaluate the model further, the authors created an experiment to generate *synthetic data* based on an input transition probability set. This step was implemented because of lack of real data available to us at this current time. Consider the simple transition probability matrix for three states in Table 3.2.

[5]The time results in this graph were from an Intel Core i7-7700K processor



(a) When MDQL with MNFQ was trained with a learning rate of 0.001.



(b) When MDQL with MNFQ was trained with a learning rate of 0.1.

Figure 3.13: Figures for the relative percent difference with training learning rates 0.001 and 0.1.

State	Simulated Results	Actual Results
Not Homeless	43620	102979
Transitional Housing	3433	1231

(a) Best results with a learning rate of 0.001.

State	Simulated Results	Actual Results
Not Homeless	106636	102979
Transitional Housing	810	1231

(b) Best prediction results with a training learning rate of 0.01.

State	Simulated Results	Actual Results
Not Homeless	100335	102979
Transitional Housing	1391	1231

(c) Best prediction results with a training learning rate of 0.1.

Figure 3.14: Figures for the prediction results of MDQL with MNFQ for each learning rate used in this evaluation.

	S1	S2	S3
S1	0.25	0.75	0.00
S2	0.00	0.25	0.75
S3	0.00	0.00	1.00

Table 3.2: A simple transition probability matrix used in the generated homelessness dataset.

This experiment assumes that this matrix is valid for all 52 weeks in the year. This probability matrix forces the population to eventually transition to state S3 and be unable to transition out of it. The weekly data generated from the experiment follows the matrix perfectly. For example, if 100 individuals are in S1 , the next week will place 25 in S1 and 75 in S2. To lengthen the time that the resulting simulation will take to converge, the experiment initially places more people in the state that is less likely to be reached. For the transition probability table in Table 3.2, with a total population of 100, this resulted in S1 starting with 59 individuals, S2 with 33, and S3 with 8. From Table 3.3, one can see that this dataset (containing 52 weeks in total) converges quite quickly. Furthermore, consider Figure 3.15 that shows the state populations over each week.

WEEK	S1	S2	S3
0	59	33	8
1	14	53	33
2	3	24	73
3	0	9	91
4	0	2	98
5	0	0	100
6	0	0	100

Table 3.3: A simple dataset generated from Table 3.2 probabilities to help further verify the model.

With the proposed model, a perfect result would be a graph that looks like Figure 3.15 after having been trained with the synthetic dataset then asked to simulate a starting population equal to week 0 in Table 3.3 for one year (52 weeks). In Figures 3.16 and 3.17, the model was run for 75 epochs with randomly initialized and provided transition probabilities

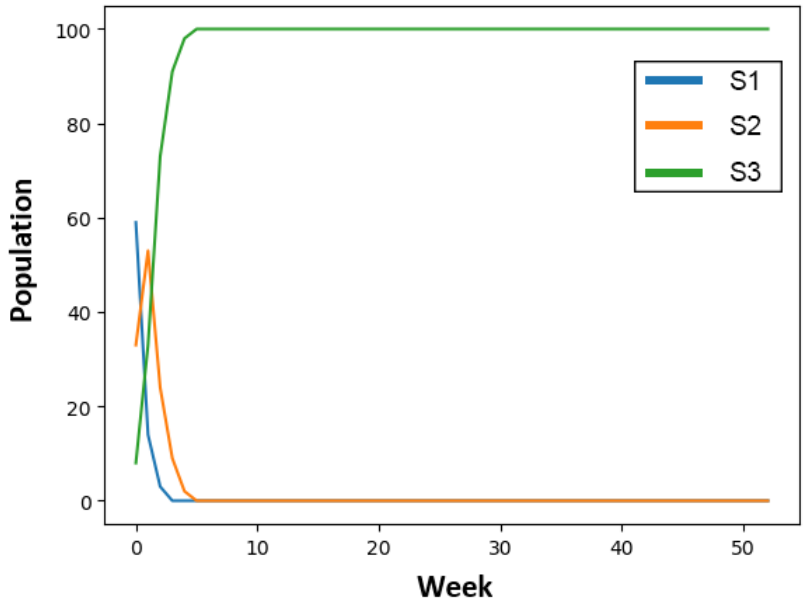
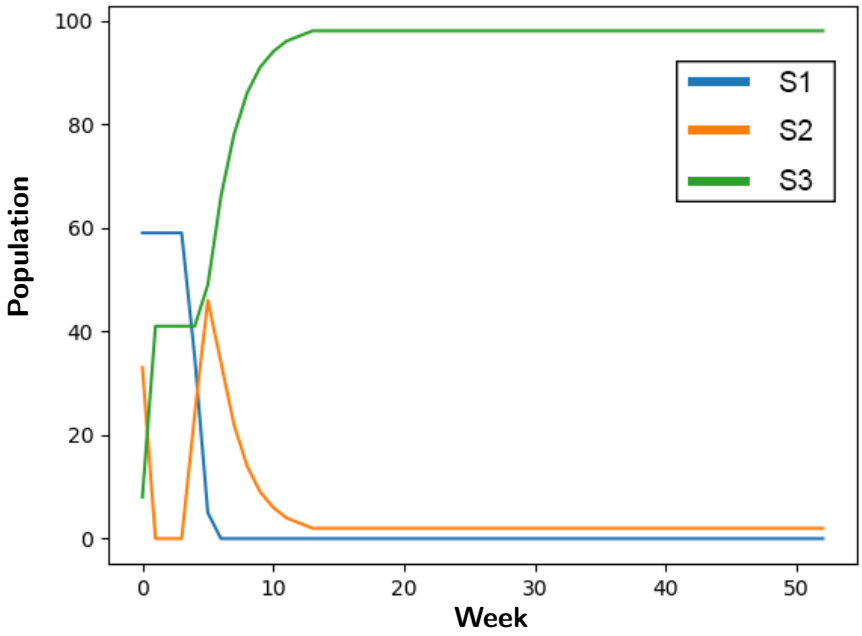


Figure 3.15: A graph of the state populations over weeks from the dataset shown in Table 3.3.

respectively. These figures also include the loss over epochs as well to show how the probability set converged as the model was trained (i.e., a lower loss means a smaller difference between transition matrices from the previous epoch).

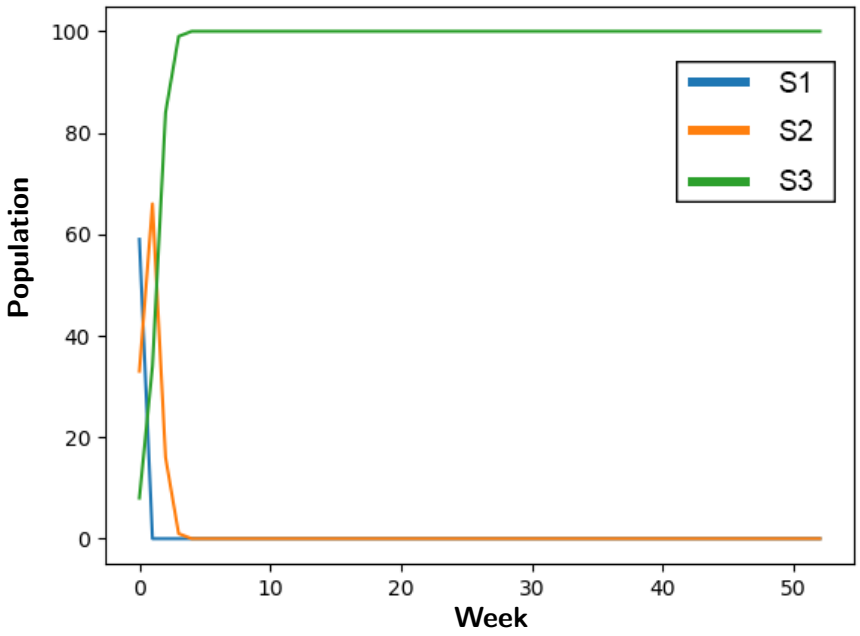
To compare the results with the previous testing, the model was ran again but with the transition probability provided to it; consider Figure 3.17. The resulting graph is very close to Figure 3.15 but has an interesting curve in the loss over epochs, which could be attributed to the fact that the model knew the exact transition probabilities (the loss dropped significantly at the start) and started to overfit the model as seen with the lack of curves and sharp lines in Figure 3.15.



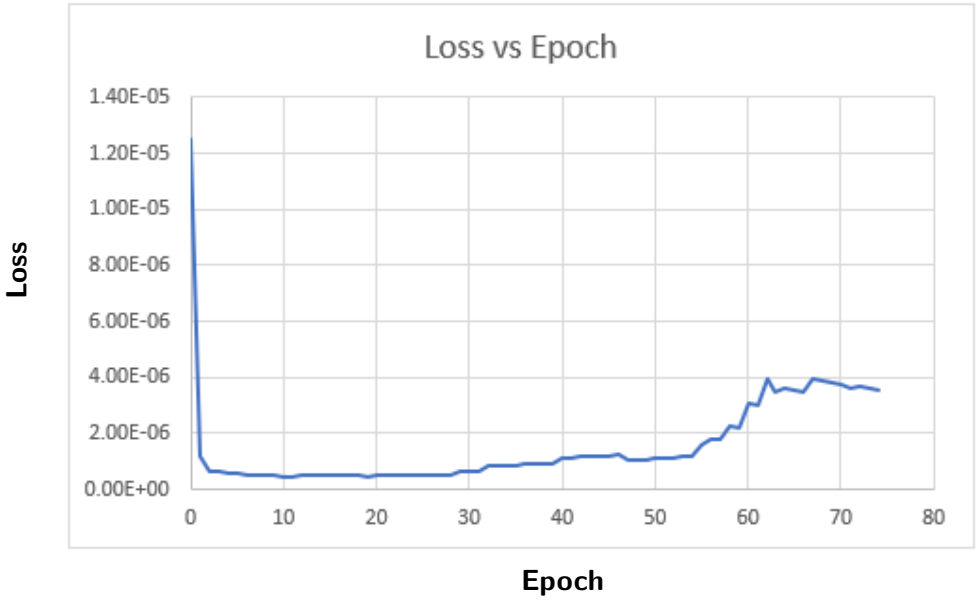
(a) The state population over time prediction of MDQL with MNFQ after initializing with random transition probabilities.

(b) The loss over epochs of the model when training on the generated dataset.

Figure 3.16: The results of a model using a learning rate of 0.01 and with initially random transition probabilities.



(a) The state population over time prediction of MDQL with MNFQ after initializing with the transition probability matrix.



(b) The loss over epochs of the model when training on the generated dataset.

Figure 3.17: The results of a model using a learning rate of 0.01 and with the transition probabilities provided to it.

3.6 Discussion and Future Works

As this work has shown, the proposed model is able to learn very quickly with a relatively small amount of data to produce accurate results. Although the test data was very low resolution, the authors are confident in the simulation as it learned from real data and was able to produce a simulation that ended with a low error to the desired results. One of the aspects that can improve here is the overall processing time. Although using CPU as the hardware accelerator was shown to be considerably quick, processing with a GPU would only speed up the algorithm even more.

One may argue that reinforcement learning may be too complex for this problem. This method is applicable where there's the need to learn simultaneously (1) the dynamics of the system, and (2) a control policy suitable for achieving some externally-imposed goals. Because the purpose of this research is to provide policy makers and planners with a means of predicting the future populations of each homelessness state, the authors feel that it is necessarily complex. It can be argued that the algorithm could indeed be used to achieve externally imposed goals based on how it is used. For example, a planner may add or remove shelters based on the output, which the algorithm will then adapt to in order to provide realistic outputs based on the real-word data on which it was trained.

Therefore, the authors disagree that using a simpler, modelling strategy (such as a MLP classification network using multi-class cross-entropy loss) would be better for this problem. Although the initial training could be considered as such, the end result is a model that will predict the population distributions over time. This could, again, be argued as classifying individuals into different population groups as a function of time but a problem arises when determining a proper function of the current state and any desired auxiliary information. By using the deep learning methods as proposed in this chapter, the algorithm is, instead,

learning this function on its own and removing that complexity which would be difficult to accurately create otherwise.

Another aspect to improve on with the model is the process of transitioning individuals to new states. For the shelter state, for example, an individual wouldn't be able to transition to it if the city they are in has no availability in their shelters. To implement this feature, the authors would suggest looking into adding a Markov decision process to replace the roulette wheel approach. By using real data from shelters in the city the user is targeting, one would get realistic populations in this example state.

As noted previously, the MDQ network takes the index of an individual's current state, as well as the individual's new state. An interesting approach would be to convert these indices to *onehot encoding* instead as they may be considered as being ranked or ordered otherwise. Consider the example shown previously where Shelter was 0, Street was 1, and Hospital was 2. For this example, these would be converted to 100, 010, and 001 respectively. From the authors' testing with the synthetic dataset described previously, the differences were quite subtle. However, to see how it would affect extreme cases, consider a model that has a high learning rate (0.1) and trains for 100 epochs. As shown in Figure 3.18, the differences are not substantial but one can see that without onehot encoding, the result is indeed closer to the exact result shown in Figure 3.15.

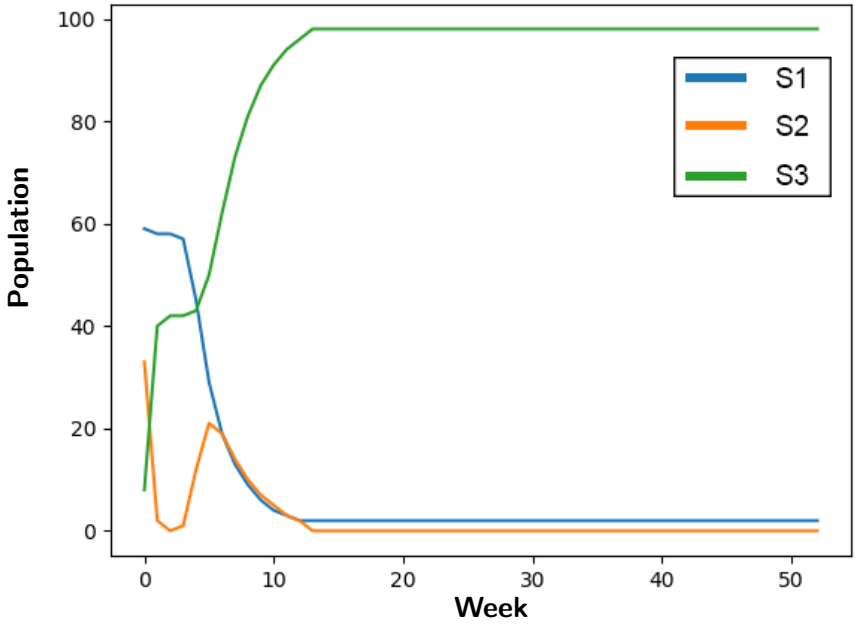
3.7 Conclusion

By creating a model to simulate a population of homeless individuals accurately, this work can provide policy makers and planners with a means of predicting the future populations of each homelessness state. If one were to simply use a mathematical model, it would be a difficult task to create as they would constantly need to revise the model manually to

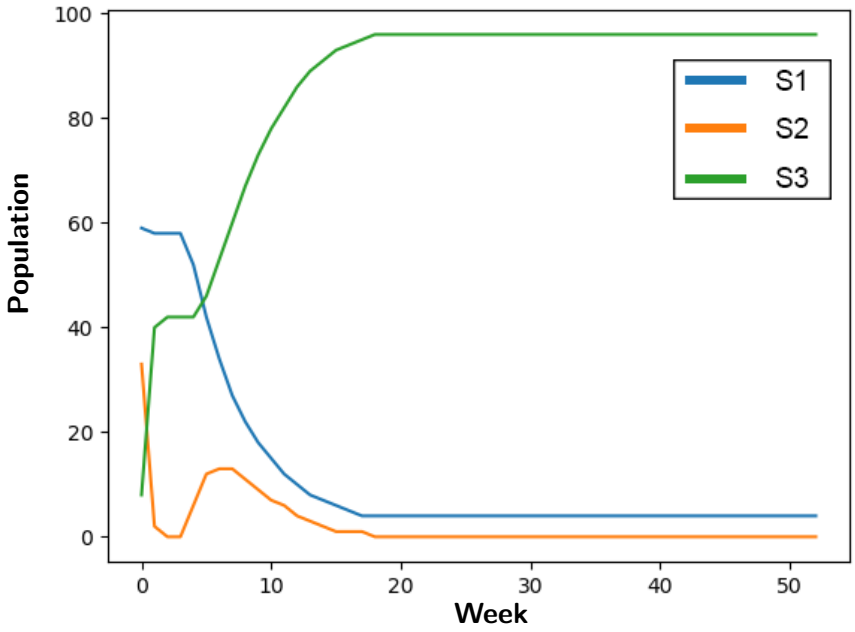
produce realistic results for each transition probability.

Instead, the authors propose a model that leverages the processing power of deep learning to achieve this result. With this model, one can input known homelessness data and have the probabilities revised dynamically to produce more accurate results. The two algorithms, modified deep q-learning and modified neural fitted q-learning, work together to achieve the same effect and input the resulting probabilities into a Markov decision process to transition the population between states.

The main challenge of this research was the lack of high resolution homelessness data, which is important as the proposed model needs to train on and produce realistic results. One way the authors accommodated this consideration was by simulating each member of the population *individually*. From the performance analysis, the authors were able to see the approach produce an accurate model with a relative percent difference of 12.5% on a low resolution data set that was entirely different from the training data set. Furthermore, with a synthetic dataset, the authors applied the algorithm to a higher quality source and confirmed that it indeed produces accurate results.



(a) The model trained on the generated dataset without onehot encoding.



(b) The model trained on the generated dataset with onehot encoding.

Figure 3.18: Models trained using a learning rate of 0.1 for 100 epochs with and without onehot encoding.

Chapter 4

BEAUT: An ExplainaBle Deep LEarning Model for Agent-Based PopUlations With Poor DaTa

All of this chapter will be submitted as the following peer-reviewed journal article:

- Fisher, A., Gajderowicz, B., Latimer, E., Aubry, T., & Mago, V. (2021). BEAUT: An Explaina**B**le Deep L**E**arning Model for **A**gent-Based Pop**U**lations With Poor Da**T**a.

This work extends the previous chapter by improving the deep learning algorithm as well as the datasets that were used. Specifically, the following improvements were implemented: training enhancements such as "random restarts" and "decaying learning rates", a neural network referred to as "adaptable parameters" which learns the population fluctuations over time, and grouping of individuals in the training dataset based on personal attributes so that the transition probability matrices being generated can take this into account. In addition to this, other "black box" models are evaluated to see how this explainable approach places. The text in this chapter will be used in a journal article that is currently being finalized.

4.1 Introduction

Time-series forecasting is an important tool for predicting the future state of a system. In many social sciences, predicting the future size of a population, its distribution, or the state of its individuals is required to create policies that can respond to changes in such population characteristics [102]. Time-series data that captures different events and trends that change population size has been a keen interest of researchers. Recent analysis into population dynamics includes extreme events [26], population viability [68], and a correlation between multiple factors like housing cost and homelessness [49]. A transition matrix is one such analysis tool as it can generate estimates of deterministic parameters such as population growth rate, sensitivities and elasticity, equilibrium population structure, and reproductive values [68]. A transition matrix is a mathematical formulation that captures the rates of change of a system from one state to another. In many social sciences, information relevant to such parameters is missing or must be adjusted when a transition matrix was trained on a population differing from the target populations to be predicted. This chapter proposes and evaluates an extension of the method presented in Chapter 3 for forecasting a population using various social services. Additionally, this method is generalized by introducing adaptable parameters to automatically accommodate different reporting and exit rates between the training and target populations.

Motivation

With the advancements in data analysis for policy evaluation, models predicting service usage and success rates in the social services domain can benefit greatly by extending data collection methods for studies addressing homelessness. The field could gain invaluable insights from evaluating the effectiveness of alternative programs before their implementation as a policy.

Data-driven policy has been successfully instrumented in recent years but is limited to certain sectors which are able to collect sufficiently detailed data, and for which suitable algorithms exist. Suitable algorithms are those that can take available data as input, and reliably produce actionable outputs that inform policymakers. For social data about vulnerable populations, a set of strict requirements must be met for data and algorithms to be actionable [42].

A primary requirement is adherence to standards governing the collection, evaluation, and deriving of policy based on data about vulnerable populations including transparency, ethics, accountability, and data-ownership [82]. A consequence of such strict requirements is the lack of available data about the individuals who use social services, especially those service clients who are experiencing homelessness. The lack of data, however, is not unique to the social service domain, but applies to any domain that includes hard to reach populations. In fact, it is detrimental to study outcomes if such populations are excluded from their analysis, as the results are less accurate when compared to studies that include hard to reach populations [97]. Several methods have been used successfully to address such issues, as discussed in section 4.1. Finally, any machine learning model producing actionable results must be explainable meaning that they must provide enough transparency to explain why a decision was made and how to adjust it for the target populations in an ethical manner. The model presented here can learn population-specific adaptable parameters during the training phase, and such weights can be easily adjusted manually to reflect the drop-off rates of a population the model was not trained on as it produces transition probability matrices. The next subsection introduces the definition of comprehensive data-driven policy and its impact on social services.

Data-Driven Homeless Policy Lacks Data

Several key issues regarding the quality of homelessness data have been identified, making it challenging to perform comprehensive data-driven policy-making in the homeless domain. Firstly, certain information is under-reported by people experiencing homelessness, especially those living with mental illness or substance abuse [98]. This is especially true for self-reported symptoms due to memory loss or social desirability bias [15, 136]. Secondly, due to the dynamic and transient nature of this population, it is difficult to track individual clients that use social services or participate in an intervention program [91, 137]. Some causes include frequent changes in address or phone numbers or an unconventional social network structure [132, 32]. Not being able to track individuals makes it challenging to capture their unique lived experiences, as well as the systemic and symptomatic factors that influence their lives. These include structural factors, coping strategies, history of physical and mental health, and spending habits [135, 22, 18]. Thirdly, many participants leave the program before completion [118], reducing the quality of longitudinal studies that need to capture complex factors over time. Many studies rely on a qualitative analysis where data can be captured as interviews, providing more depth of information about those clients that remain [13, 43, 95, 104]. However, without a considerable amount of thematic analysis to evaluate and code the data, date-driven methods lack the structure to interpret such results [94]. Finally, point-in-time counts are a popular methodology for collecting data about homeless populations [49]. They provide a snapshot of the count of different demographics, as well as their geographical and seasonal distribution [92]. This method provides descriptive analysis, giving insights into trends over time such as housing availability, and capture information about the hardest to reach- mainly chronically homeless.

Several outreach strategies have been used to increase enrolment and retention rates, but

these methods are difficult to transfer between target populations [37, 59, 91]. Some success has been observed with tracking transient population through web-based methods like social networks, on-line telephone and address directories, as well as public record resources such as judicial and death records [134]. Some strategies address data reliability by actively working towards increasing participation and retention. For example, participation was increased in a longitudinal study by providing participants experiencing homelessness access to computer facilities during the study [30].

4.2 Related Work

Data-Driven Policymaking

In an era of smart homes and smart cities, Bibri highlights three points of focus for data-driven policies that impact urban centres of the future [8]. These include the use of technological advancement towards sustainable development through optimal processes, the potential of big-data technology, and describes a novel architecture for data-driven smart cities. Big-data especially provides an opportunity for policymakers to evaluate alternative policies based on empirical evidence and prediction models trained on available data [106]. These policies are particularly challenging in a social context due to the complex nature of social systems. Tsoukias et al. present an architecture that outlines analytics performed within a policy cycle of design, testing, implementation, evaluation, and review of public policies [128].

Given any such algorithmic approach as a viable solution, policymaking will raise ethical questions that must be addressed, including moral consequences, stakeholder rights and responsibilities, decision delegation, and accountability [90, 121]. Bertsimas and Kallus suc-

cessfully combine machine learning with operations research and managerial science methods to infer optimal policy decisions from data [7]. While work by Bertsimas and Kallus demonstrates the potential to perform prescriptive analysis, the application domain, namely the inventory management problem, does not face the types of ethical issues presented in this chapter. Aziz et al. demonstrate optimal polices in an ethically sensitive domain, prioritizing homeless youth for housing resources by allowing a policymaker to configure a *fairness index* [24]. Relying on mixed-integer optimization, this work claims to produce fair, efficient, and explainable policies according to this index. Piscopo et al. rely on machine learning to identify dimensions required to implement a policy targeted at a given community successfully [101]. Here, the random forest algorithm is used to identify optimal levels of sense of community and participation following the implementation of a given policy.

Time-Series Forecasting

Many forecasting methods have been developed that rely on the perceived dependency of observed time-series data to predict future data. Thse include Autoregressive Integrated Moving Average (ARIMA) [93], Gated Recurrent Unit (GRU) [93], Long-Short Term Memory (LSTM) [57], and Recurrent Neural Networks (RNN) [99]. Traditional regression-based classification methods have also been observed to successfully forecast time-series data, including Random Forest and network Multi-Layer Perceptrons (MLP) [1, 65]. A comparison of various machine learning methods shows that a simple regression based MLP has the best results overall [1]. In fact, a simple neural network still performed as well as time-series specific methods such as ARIMA [115]. Time-series data can exhibit a number of characteristics which require a customized modelling methods. For example, Ding et al. evaluate extreme events captured by time-series data, and propose a new loss function along with

a memory-based neural network [26]. Unlike GRU and LSTM models that use memory to remember the sequence of data points, the memory module developed by Ding et al. remembers extreme events and incorporates them into forecasting future events. Based on the analysis of existing methods, it is clear that regression-based methods are one of the most appropriate to perform time-series forecasting. The model presented in this chapter takes advantage of regression with simulation of individual agents.

4.3 Preliminaries

Evaluation Data

The algorithm is evaluated using real-life time-series data capturing the state of ~ 937 individuals from 2009 - 2013 [80], where the state is one of the following: street, shelter, hidden homeless, not homeless, transitional housing, and institution (i.e., hospital, rehabilitation, and prison). The data was collected as part of the AHCS study, a multi-site randomized controlled trial of the Housing First intervention program [50]. The population selection focused on people living with mental illness who were homeless or had recently been homeless but were precariously housed at the time of recruitment. Of the 2,866 people assessed for eligibility across five Canadian sites between 2009 and 2013, 2,225 were accepted into the study. Of these, 1,265 were placed in the main AHCS experimental group (Housing First or "HF") and 990 in the Treatment-As-Usual (TAU) control group.

The algorithm presented in this chapter is evaluated using additional data collected for this control group from a questionnaire administered to these 990 participants called the Residential Follow-Back Timeline (RTLFB) [126]. Out of these response, 937 participants provided usable answers (94.6%). The purpose of the RTLFB questionnaire is to collect

detailed information about participants, including their housing situation, the number of changes between states in a given period, the reason for moves, and the type of residence. The format of the RTLFB questionnaire administered as part of the AHCS study underwent minor adaptations for the Canadian context [80]. RTLFB was a self-reported study, administered every 3-months, asking participants to recount their situation each day after the last time RTLFB was administered. The evaluation presented here is based on the usable RTLFB answers for 937 participants, aggregated at weekly intervals. Furthermore, the individuals are spanned across the 2 groups (TAU or HF) within 12 subgroups based on age, homelessness history, and gender for a total of 24 sub-populations as follows:

- **Age:** Under 30, Between 30 to 50, or Greater than 50
- **Homelessness History:** Homeless less than 1 year or homeless more than 1 year
- **Gender:** Male or female

Forecasting Populations

In this subsection, notation for forecasting future populations that use various services in a community or are in a specific housing situation will be presented. Let S be a set of possible states an individual can be in and the length of this set be $I = |S|$ where $I \, \epsilon \, \mathbb{R}$. A state within the set S is defined as s_i where the index $i \, \epsilon \, [0, I)$. For example, when simulating homeless individuals, the set of states may be $S = [street, shelter, hospital]$ which would mean that $I = 3$ and $s_0 = street,\ s_1 = shelter,\ s_2 = hospital$. The total number of time periods in a simulation is defined as T where $T \, \epsilon \, \mathbb{R}$ and a specific time period is defined as t where $t \, \epsilon \, [0, T)$. For example, the time period may be in weeks where $T = 52$ which would mean that the first week would be $t = 0$. For a specific time period t, the total

number of individuals in a certain state i in the real dataset is defined as $\mathbf{n}_{i,t}$ and the total number of individuals overall is defined as $\mathbf{N}_t$. These values as predicted by the algorithm is defined as $\hat{n}_{i,t}$ where $\hat{n}_{i,t} \epsilon \mathbb{R}$ and $\hat{N}_t = \sum_{i=0}^{I} \hat{n}_{i,t}$ respectively. To refer to each individual in this population at time period t, let $a_{p,t}$ be an individual with a unique index number p. The set of all individuals in a simulation is defined as A_t. Additionally, an individual's state at time period t is defined as $s_{p,t}$ and the set of individuals in state i is defined as $c_{i,t} = \{a_{p,t} : s_{p,t} == i\}$ where $|c_{i,t}| = n_{i,t}$. To recap, Table 4.1 shows a list of notation discussed so far.

Symbol	Definition
S	A set of states
I	The total number of possible states which is equal to $\|S\|$ where $I \epsilon \mathbb{R}$
s_i	State i in the set S where $i \epsilon [0, I)$
T	The total number of time periods in the simulation where $T \epsilon \mathbb{R}$
t	The current time period in the simulation where $t \epsilon [0, T)$
$\hat{n}_{i,t}$	The predicted number of individuals in state i at time period t where $\hat{n}_{i,t} \epsilon \mathbb{R}$
$\mathbf{n}_{i,t}$	The actual population in state i at time period t from the dataset
$\hat{N}_t$	The predicted total number of individuals at time period t which is equal to $\sum_{i=0}^{I} \hat{n}_{i,t}$
$\mathbf{N}_t$	The actual total number of individuals at time period t in the dataset
A_t	A set of the individuals at time period t where $\|A_t\| = \hat{N}_t$
$a_{p,t}$	An individual a, indexed by their unique ID $p \epsilon \mathbb{N}_0$, at time period t, in the set A_t. This contains characteristics of the individual including current state ($state(a_{p,t})$) and length of stay ($length(a_{p,t})$) in their current state

$s_{p,t}$	An individual p's state at time t which is equal to $state(a_{p,t})$
$c_{i,t}$	The set of the individuals in state i at time t which is defined by $\{a_{p,t} : state(a_{p,t}) == i\}$. Additionally $\|c_{i,t}\| = n_{i,t}$

Table 4.1: A recap of the notation discussed in section 4.3

Time-Series Forecasting: An Explainability Problem

One of the drawbacks of traditional machine learning methods applied to time-series forecasting is that they are essentially a black box– an opaque model that can not clearly– or in domain-specific terms, explain why it has made the predictions it has [106]. Explainability is especially important when applying forecasting methods to data-driven policymaking where decisions must be transparent, ethical, and allow for accountability [82], or when indicators show unexpected results [69] or shifts in data that seem random [2]. A popular method for revealing the algorithm's decisions is to model the underlying factors producing time-series data as a set of discrete, identifiable, and explainable events. A sequence of such events, when chained together in some configuration, will then produce the observed time-series data. To generalize such events, a transition matrix can be derived with which to simulate the transition from one event to another over time. A method such as Markov decision process [139, 6] can then be used to simulate the transition from one state at time t to another state at time $t + 1$.

In the BEAUT algorithm, transition probability matrices are created that define the probability an individual will transition from one state to another. These are defined for each time period t as Q_t and allow for the condition $t \epsilon [0, T)$ to be flexible based on the implementation. For example, in the case of T being a number of weeks, it may be desired

that there be a matrix for each week of the year. In this case, the appropriate matrix would be defined as $t = t \ \% \ 52$. Therefore, if $t = 52$, the simulation would be referring to the first week of the second year so the first week's matrix would be selected[1]. Within this matrix, there is a probability for each pair of states in S such that the matrix Q_t is of size $I \times I$. An individual probability for time period t within matrix Q_t is defined as $Q_{i,j,t}$ when transitioning from state i to state j where $j \ \epsilon \ [0, I)$.

4.4 Forecasting Time-Series with BEAUT

Methodology

Algorithm 1 presents a basic overview of the homelessness forecasting, and Figure 4.1 is a visual representation. First, the overall population A and the current week in the year t are passed to the model. The population is simply an array of individuals indexed by $p, t \in A$ that each represent an individual as denoted by $a_{p,t}$ at time period t. Next, the appropriate transition probability matrix for this week in the year is retrieved (tr_mat on line 2), and three empty arrays (lines 3 - 5) are initialized to keep track of the total transitions between states (tr_counts), the best transition probabilities the modified deep Q-learning (MDQL) network outputs (mdq_probs), and the total rewards from the modified neural fitted q-iteration (MNFQ) networks for each transition respectively ($mnfq\_rewards$).

Next, the model starts by cycling through each individual p in the population (line 6). First, their current state is recorded ($s_i = s_{p,t}$ on line 7), then a new state for the individual is determined (s_n on line 8) from the roulette wheel ($Wheel$) approach using the transition probabilities from their current state ($tr\_mat[s_i]$) as defined in Algorithm 2. The new state

[1] $t = 52 \ \% \ 52 = 0$

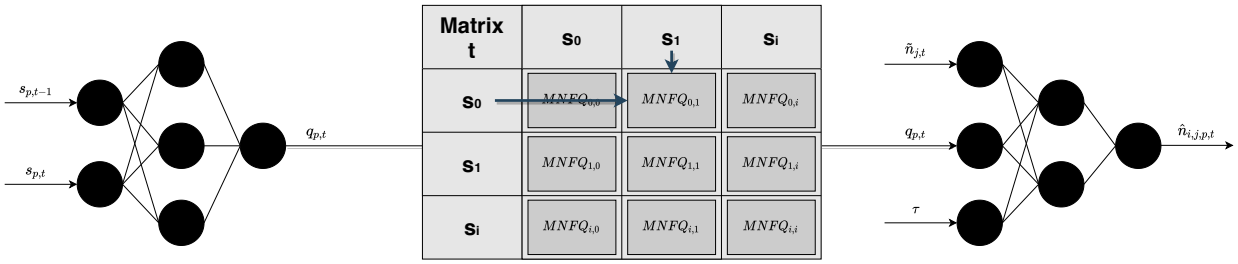


Figure 4.1: An overview of BEAUT's layout which shows, from left to right, the MDQL network, a transition probability matrix, and a MNFQ network for an individual transition probability.

for the individual is then stored ($s_{p,t+1} = s_n$ on line 9). Next, the current and new states are processed through the MDQL network ($mdq\_nn.process(s_i, s_n)$ on line 10) to get the output probability ($q_{p,t}$) for individual p. This is then processed through the MNFQ network for the new state s_n (line 11), along with total population count in the new state (line 12) to get the reward ($\hat{n}_{j,p,t}$ used on line 13) and whether or not the *curr_mdq_prob* produces the *lowest_error*. If this is true (line 14), then the *curr_mdq_prob* is stored in the *mdq_probs* matrix for this transition (line 15).

After all individuals in the population have been processed, the *mnfq_rewards* are averaged for each state-to-state transition based on the *tr_counts* (line 19). Then, the *new_tr_mat* is calculated by updating the current *tr_mat* based on the *mdq_probs* and *mnfq_rewards* (line 20). Next, the *loss* is determined by getting the *mean_difference* between the *new_tr_mat* and the current *tr_mat* (line 21). Lastly, the current transition matrix is updated to the new one (line 22) and the *loss* is returned from the algorithm.

MDQL

As discussed in previous chapters, when training modified deep q-learning (MDQL) as a standalone algorithm, for each time period t, each individual $a_{p,t}$ is simulated by a neural network by considering their previous state $s_{p,t-1}$ and new state $s_{p,t}$ to output a transition probability $q_{p,t}$ for this individual and their transition. This value is considered an "ideal" probability in the sense that, based on all previous transitions that have went through this network, this probability makes the most sense for this particular transition.

To provide a detailed overview of this extended approach, notation will be defined for every step of the algorithm and may overlap with Chapter 3 in some aspects. Once an individual has passed through this network, their output is compared with the actual transition probability for $Q_{s_p,s'_p,t}$ where $s_p = s_{p,t-1}$ and $s'_p = s_{p,t}$ to determine a "reward" for the network. This is defined as follows:

$$r_{p,t} = q_{p,t} - Q_{s_p,s'_p,t} \tag{4.1}$$

Once all individuals in A_t have been processed, sets of individuals that transitioned from state i ($c_{i,t}$) to state j ($c_{j,t+1}$) are created by taking their intersection defined as $c^{\cap}_{i,j,t} = c_{i,t} \cap c_{j,t+1}$. The size of each of these sets $|c^{\cap}_{i,j,t}|$ is then used in the formulae to update each transition probability $Q_{i,j,t}$ as follows:

$$Q_{i,j,t} = Q_{i,j,t} + [1 + \frac{\sum\limits_{p \epsilon c^{\cap}_{i,j,t}} r_{p,t}}{|c^{\cap}_{i,j,t}|}]\eta \tag{4.2}$$

After each probability has been updated, the following error is calculated to be used for back-propagation on the MDQL network:

$$E_t = \frac{\sum_{i=0}^{I} \sum_{j=0}^{I} [Q_{i,j,t} - \frac{\sum_{p \epsilon c_{i,j,t}^{\cap}} q_{p,t}}{|c_{i,j,t}^{\cap}|}]}{I^2} \tag{4.3}$$

To recap, Table 4.2 shows the notation discussed in this section.

Symbol	Definition
Q_t	The transition probability matrix for time t
$Q_{i,j,t}$	A transition probability in the current matrix Q_t for going from state i to state j where $i,\ j\ \epsilon\ [0, I)$
$q_{p,t}$	The output of the MDQL network for person p at time period t. This is based on the individual's previous state $s_{p,t-1}$ and their current state $s_{p,t}$.
$r_{p,t}$	The reward for individual p at time period t when just using MDQL.
$c_{i,j,t}^{\cap}$	A set of the individuals that transitioned from state i to state j at time t. This is equal to the intersection $c_{i,t}\ \cap\ c_{j,t+1}$ and the size is equal to $\|c_{i,j,t}^{\cap}\|$
E_t	The loss measure for MDQL at time t

Table 4.2: A recap of the notation discussed in section 4.4

Combining MDQL+MNFQ

When using modified neural fitted q-learning (MNFQ) with MDQL, for each time period t, each individual $a_{p,t}$ is processed through the MDQL network as defined before. However, before the reward is calculated, each individual is processed through an MNFQ network for

this state i to state j transition defined as the $MNFQ_{i,j}$ network. These networks consider the individual's output from the MDQL network $q_{p,t}$, the current population in state j at time t as of the individual's transition $\tilde{n}_{j,t}$, and the number of time periods until the next data instance τ to output what the end population in state j will be once the time period τ has elapsed; this output is defined as $\hat{n}_{i,j,p,t}$.

Once an individual has been processed through the MNFQ network for their transition, the error formula defined before is re-purposed as follows:

$$E^*_{j,p,t} = \mathbf{n}_{j,t} - \hat{n}_{i,j,p,t} \tag{4.4}$$

This error is then used in a re-purposed rewards formula defined as follows:

$$r^*_{p,t} = (\frac{q_{p,t}}{\sum_{i=0}^{I} Q_{i,j,t}})(\frac{E^*_{i,j,p,t}}{\mathbf{n}_{j,t}}) \tag{4.5}$$

The training for MNFQ networks comes from individuals that give a low error $E^*_{j,p,t}$ as defined in Equation 4.4- if it is 20% or less, back-propagation is performed. Furthermore, for each transition from state i to state j, the individual p that gives the lowest error is stored as individual m. Once all individuals have been processed, an "offset" for each transition probability $Q_{i,j,t}$ is calculated using individual m as follows:

$$\Delta Q_{i,j,t} = \frac{q_{m,t}}{\sum_{k=0}^{I} Q_{k,j,t}}(\frac{E^*_{i,j,m,t}}{\mathbf{n}_{j,t}}) \tag{4.6}$$

With this offset, the probability update formula defined before is re-purposed as follows:

$$Q_{i,j,t} = Q_{i,j,t} + [\frac{\sum_{p \epsilon c^{\cap}_{i,j,t}} r^*_{p,t}}{|c^{\cap}_{i,j,t}|}]\eta + \Delta Q_{i,j,t}\eta \tag{4.7}$$

Once all probabilities have been updated, the error of the output neuron on the MDQL network is set to the average difference in the updated transition probabilities to that of the

original ones to perform back propagation with. This process is repeated for all time periods t in the set $[0, T)$ to train the algorithm on a given dataset. When the training process has completed, a set of populations for each state in S can be input along with the desired time period T to simulate the population using the generated transition probability matrices. To recap, the notation discussed in this section can be seen in Table 4.3.

Symbol	Definition
$\hat{n}_{i,j,p,t}$	The output of the MNFQ network for the transition from state i to j ($MNFQ_{i,j}$) for individual p at time t. This is based on individual p's output from the MDQL network ($q_{p,t}$), the current population in state j at time t as of this transition ($\tilde{n}_{j,t}$), and the number of time periods until the next data instance (τ).
τ	The number of time periods until the next instance in the dataset
$E^*_{i,j,p,t}$	The error used in MNFQ's rewards and updates formulae.
$r^*_{p,t}$	The rewards formula used by MNFQ for individual p at time period t.
m	The p of the individual that produced the lowest error $E^*_{i,j,p,t}$ when transitioning from state i to j.
$\Delta Q_{i,j,t}$	The offset for $Q_{i,j,t}$ calculated from the individuals that transitioned from state i to j through the $MNFQ_{i,j}$ network.

Table 4.3: A recap of the notation discussed in section 4.4

Adaptable Parameters

To further refine the algorithm so it closely follows the dataset, this chapter introduces *adaptable parameters*. These are neural networks that consider a set of inputs and output a value between 0 and 1. For the purposes of simulating a population of individuals, the two most important applications of this are for **exiting** and **spawning** individuals to the simulation. Since the algorithm defined before just simulates each individual in the population, these changes cannot be done otherwise.

For exiting individuals, the adaptable parameter will be referred to as AP_x. First, the training process described in Algorithm 1 is completed for a time period t. Then, before beginning the process for time period $t+1$, it calculate the percentage of individuals that would need to exit the simulation in order to equal the total population in the real dataset as follows:

$$x_t = \frac{\hat{N}_t - \mathbf{N}_t}{\hat{N}_t} \tag{4.8}$$

The adaptable parameter AP_x tries to produce this result by taking the simulated population size $\hat{N}_t$ and time period t as inputs to produce a percentage to exit $\hat{x}_t$. The difference between $\hat{x}_t$ and x_t is then used to perform back propagation on the network. Once the training has completed, this parameter is used at the end of each time period t to remove a percentage of the population from each state population $\hat{n}_{i,t}$ proportionately before beginning the next time period $t+1$.

For spawning individuals, this parameter will be referred to as AP_z. An identical process is followed as described for AP_x but the percentage of individuals that would need to spawn into the simulation in order to equal the total population in the real dataset is defined as follows:

$$z_t = \frac{\mathbf{N}_t - \hat{N}_t}{\mathbf{N}_t} \tag{4.9}$$

This is then compared against the output of AP_z, defined as $\hat{z}_t$, to perform back-propagation on similar to AP_x. Again, once training has completed, this parameter is used after each time period t but this time to spawn individuals into each state proportionately. A recap of the notation discussed in this section can be seen in Table 4.4.

Symbol	Definition
AP_x	The adaptable parameter for *exiting* individuals. The output of this is $\hat{x}_t$ where $\hat{x}_t \epsilon [0, 1]$ and it is based off of the size of the total number of individuals at time t ($\hat{N}_t$) and the current time period t.
x_t	The percentage of individuals that need to exit from the simulation to equal the actual $\mathbf{N}_t$ population at time t. The difference between $x_t - \hat{x}_t$ is used to train the adaptable parameter AP_x
AP_z	The adaptable parameter for *spawning* individuals. The output of this is $\hat{z}_t$ where $\hat{z}_t \epsilon [0, 1]$ and it is based off of the size of the total number of individuals at time t ($\hat{N}_t$) and the current time period t.
z_t	The percentage of individuals that need to spawn into the simulation to equal the actual population $\mathbf{N}_t$ at time t. The difference between $z_t - \hat{z}_t$ is used to train the adaptable parameter AP_z

Table 4.4: A recap of the notation discussed in section 4.4

Assume the simulation will have 100 individuals across three states as follows $[60, 10, 30]$. Assume that the x_t parameter determined that, for the current simulation week, 1% (0.1) of

the population should exit. The first step is to normalize the state populations; this example would appear as $[0.6, 0.1, 0.3]$. Next, the total number of individuals that need to exit are determined by simply multiplying x_t by the total population ($100 * 0.1 = 10$). Next, for each state population, their normalized value is multiplied by the total number of individuals that need to exit in order to determine how many to remove from that state; this example would appear as $[0.6 \times 10, 0.1 \times 10, 0.3 \times 10] = [6, 1, 3]$. Therefore, the final populations for this example would be $[60 - 6, 10 - 1, 30 - 3] = [54, 9, 27]$. The layout of this process can be seen in Figure 4.2.

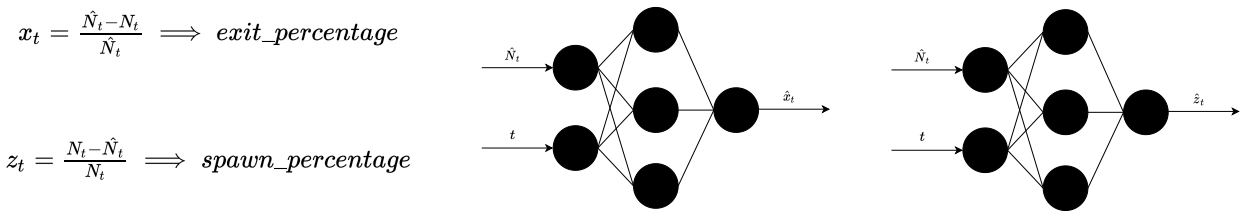


Figure 4.2: An overview of adaptable parameters' training process layout.

Evaluation Process

To evaluate BEAUT's ability to forecast homelessness populations, 2 models (i.e., one each for TAU and HF) are first trained on the At Home/Chez Soi project as described in section 4.3. Then, with this trained model, a 2015 Point-In-Time (PIT) count from the region done on March 24, 2015 is used as an input (spanned across all 24 sub-populations as defined previously and further described in section 4.5) and simulated for roughly 3 years to reach another count (further described in section 4.5) conducted on April 24, 2018 (i.e., $T = 159$) which **only** has populations for each 12 subgroups for age, homelessness history, and gender (i.e., they are not split across TAU nor HF). Therefore, with these final 24 sub-populations

from BEAUT, the 12 subgroups in both the TAU and HF outputs are first combined then, to protect confidentiality of the datasets but still provide results, all male subgroups and female subgroups are combined to finally get 2 sub-populations. These counts are shown in Tables 4.5 and 4.6, respectively, where one can note that the "Hidden Homeless" and "Not Homeless" counts are not defined. This is due to the fact that it is hard to get an accurate count for these two states but it is still one that is present in the training dataset. So, when the model is simulating from 2015 to 2018, it will still transition individuals in and out of these states but when looking at the final result, the individuals in these populations will **not** be considered.

	Male	**Female**
Street	400	30
Shelter	927	139
Hidden Homeless	-	-
Not Homeless	-	-
Transitional Housing	478	562
Institution	225	56
	2030	**787**

Table 4.5: A summarization of the 2015 PIT count.

	Male	**Female**
Street	610	67
Shelter	791	120
Hidden Homeless	-	-
Not Homeless	-	-
Transitional Housing	810	455
Institution	282	12
	2493	**654**

Table 4.6: A summarization of the 2018 PIT count.

Example Predicted Output (Values)			
	Street (S0)	**Shelter (S1)**	**Institution (S2)**
Male (M)	15	30	10
Female (F)	10	50	15

Table 4.7: An example predicted forecast from BEAUT.

Example Actual Output (Values)			
	Street (S0)	**Shelter (S1)**	**Institution (S2)**
Male (M)	10	35	5
Female (F)	5	40	20

Table 4.8: An example of the actual data BEAUT was to forecast.

To evaluate this final result, consider the simplified example predictions and actual values in Tables 4.7 and 4.8 where this combination has already been done. These populations will first be expressed as percentage distributions using the following formula:

$$P_{r,c} = \frac{v_{r,c}}{\sum_i^c v_{r,i}} \times 100 \tag{4.10}$$

where r is a row (i.e., **M**ale or **F**emale), c is a column (i.e., S**0**, S**1**, or S**2**), and $v_{r,c}$ is the value in the table at row r column c. For the formulae to follow, distinguishing between the actual value and the predicted value will be denoted with a and p respectively. After applying this to the datasets in Tables 4.7 and 4.8, it would result in distributions as shown in Tables 4.9 and 4.10.

Example Predicted Output (Percentage)			
	Street (S0)	**Shelter (S1)**	**Institution (S2)**
Male (M)	27.28%	54.55%	18.18%
Female (F)	13.33%	66.67%	20.00%

Table 4.9: An example predicted percentage distribution from BEAUT.

With the final results– values and percentages– the Weighted Mean Absolute Error (WMAE) between the predicted results and actual results, for each gender, will be calculated using the formula shown in Equation 4.11:

$$WMAE = \frac{\sum_{s=0}^{S-1} |a_s - p_s| \times a_s}{\sum_{s=0}^{S-1} a_s} \tag{4.11}$$

where S is the total number of states indexed by s. Finally, for the percentage distributions' genders' WMAE **only**, the results are further weighted by multiplying them by how many

Example Actual Counts (Percentage)			
	Street (S0)	**Shelter (S1)**	**Institution (S2)**
Male (M)	20.00%	70.00%	10.00%
Female (F)	7.69%	61.54%	30.77%

Table 4.10: An example of the actual percentage distribution BEAUT was to forecast.

individuals are in the actual output as shown in Table 4.8 where there are 50 males and 65 females for a total of 115 individuals. Therefore, the weight for males (w_M) would be $(50/115) \times 100 = 43.5\%$ and for females (w_F) would be $(65/115) \times 100 = 56.5\%$. With these final WMAEs, the process will present two plots to show (1) the total WMAE for values (i.e., the summation of the WMAE for Male and Female values) and (2) the total WMAE for percentage.

4.5 Data Enhancements

As stated in the evaluation process, the 2015 PIT count is split amongst TAU and HF to get a total of 24 sub-populations. However, the data was not originally like this and was, instead, only split amongst the 12 sub-populations based on age, homeless history, and gender. Since the training data from the AHCS study does include data points for all 24 sub-populations, it would be beneficial to be able to accurately apply this learning when

evaluating the performance of the model's ability to forecast homeless populations from 2015 to 2018. In addition to this, although the implementation of adaptable parameters is present to account for fluctuations in the population, this is **only** accurate to those present in the training dataset which was the result of a study following sets of individuals. When look at the PIT counts from 2015 to 2018, the population steadily increases simply as result of the change in time periods. Therefore, the ability to logically add individuals into the population as the simulation progresses would be desirable.

2015 PIT Count Enhancement

In order to further spread this dataset amongst the TAU and HF populations, reports from the Mouvement pour mettre fin à l'itinérance (MMFIM or "Movement to end homelessness") were requested to get an estimate of how many individuals entered housing first programs between the two PIT counts in question. Due to confidentiality reasons, the full report cannot be disclosed but rather summary statistics retrieved from it that are relevant to this work will be shown.

In total, relevant to the count, there were 369 males and 207 females that entered housing first between these two time periods. Table 4.11 shows the age distribution of these individuals and Table 4.12 shows the distribution of each individual's length of homelessness before entering the program. As stated in the introduction, the three age ranges are (1) under 30, (2) between 30 to 50, and (3) greater than 50. For the homelessness history, that is split amongst two conditions being (1) homeless less than 1 year and (2) homeless more than 1 year. From the age distribution in Table 4.11, since the range "30 to 54 Years" does not fall perfectly into the training dataset's "between 30 to 50", one could simply split the percentage stated there amongst the 24 year range to get $53\%/24\ Years = 2.2\%\ per\ year$

	Under 18 Years	18 to 29 Years	30 to 54 Years	55 to 64 Years	Over 65 Years
Men	0%	10%	53%	28%	9%
Women	0%	10%	57%	22%	11%

Table 4.11: The age distribution of individuals that entered housing first between the 2015 to 2018 PIT counts in Montréal.

so that $4\ years \times 2.2\% = 8.8\%$ could be subtracted from this distribution and added to the next distribution (i.e., greater than 50) to more closely follow the subgroups in the training population. With these statistics, the 2015 PIT count can then be split into either TAU and HF to get the resulting 24 sub-populations.

	6-12 Months	12 - 24 Months	2 to 5 Years	5 to 10 Years	Over 10 Years
Men	38%	15%	24%	18%	5%
Women	41%	14%	33%	9%	4%

Table 4.12: The homelessness history distribution of individuals that entered housing first between the 2015 to 2018 PIT counts in Montréal.

2015 to 2018 Population Increases

As the model simulates from 2015 to 2018, it would be desirable to provide a means of entering individuals into the population to emulate the real-world population change over time. To get an estimate of how to do this, questionnaires done by homeless individuals during the 2018 PIT count who had been homeless for no more than 1 year were analyzed. As a result, an estimated distribution for both the TAU and HF populations were derived to add into the overall population at the end of every month. This information is confidential and cannot be fully shared in this chapter but summary statistics will still be presented.

Table 4.13 shows the estimated distribution for monthly increases in the TAU population for both male and female, while Table 4.14 shows this information for the HF population. From further analysis on the both the PIT counts and questionnaires, the resulting populations that were split up amongst this distribution, per month, was **50** for TAU and **32** for HF. Overall, this means that the model will enter **82** new individuals, per month, while simulating from 2015 to 2018.

	Hidden Homeless	Institution	Shelter	Transitional Housing	Street	**TOTAL**
			Male			
Less than 30	5.89%	0%	5.89%	11.76%	2.94%	**26.48%**
30 to 50	2.94%	2.94%	14.71%	17.65%	5.88%	**44.12%**
51+	2.94%	0%	11.76%	11.76%	2.94%	**29.40%**
			Female			
Less than 30	6.25%	6.25%	6.25%	31.25%	0%	**50.00%**
30 to 50	6.25%	12.50%	0%	18.75%	0%	**37.50%**
51+	0%	0%	6.25%	6.25%	0%	**12.50%**

Table 4.13: An estimated distribution for monthly increases in the TAU population from 2015 to 2018.

	Hidden Homeless	Institution	Shelter	Transitional Housing	Street	**TOTAL**
			Male			
Less than 30	4.35%	0%	4.35%	13.04%	4.35%	**26.09%**
30 to 50	4.35%	4.35%	13.04%	17.39%	4.35%	**43.48%**
51+	4.35%	0%	8.69%	13.04%	4.35%	**30.43%**
			Female			
Less than 30	11.11%	11.11%	0%	33.33%	0%	**55.55%**
30 to 50	0%	11.11%	0%	22.22%	0%	**33.33%**
51+	0%	0%	0%	11.11%	0%	**11.11%**

Table 4.14: An estimated distribution for monthly increases in the HF population from 2015 to 2018.

4.6 Results

Grid Search

To determine the best parameter configuration for BEAUT, a *horizon* error was introduced by excluding the last **10** weeks of training data to see how the model would forecast the end of the dataset. This resulted in a total of 1,675 configurations being tested where it looked at the percent accuracy in this error over the epochs. To provide insight into the most influential parameters, consider Figure 4.3. The poorer line was configured with the highest MDQL learning rate out of all configurations of 0.1, as well as the lowest MNFQ back-propagation error threshold of 20%. Similarily, the constant line was configured with the same MNFQ threshold but the smallest learning rate of 0.0001. To demonstrate the sensitivity of these parameters over epoch iterations, the sensitive line was configured with a MDQL learning rate of 0.1 but the MNFQ error threshold was instead set to 50%. Compared to the poorer line which had the same MDQL learning rate, an observation can be made showing that the runs started at the same error but, around epoch 10, began to vary drastically. Lastly, the best performing model was configured with the same MNFQ error threshold as the senstive run (50%) but as for the MDQL learning rate, it was set to 0.01 and resulted in a $\sim 9.8\%$ error very quickly before going up to a $\sim 10.5\%$ error towards the end.

Performance

To properly evaluate the performance of BEAUT, the following three models were also evaluated by following the same process previously described for training and testing: (1) neural network, (2) random forest, and (3) random forest regressor. Across these three models, a total of 2,040 different configurations were ran that first trained on the AHCS dataset

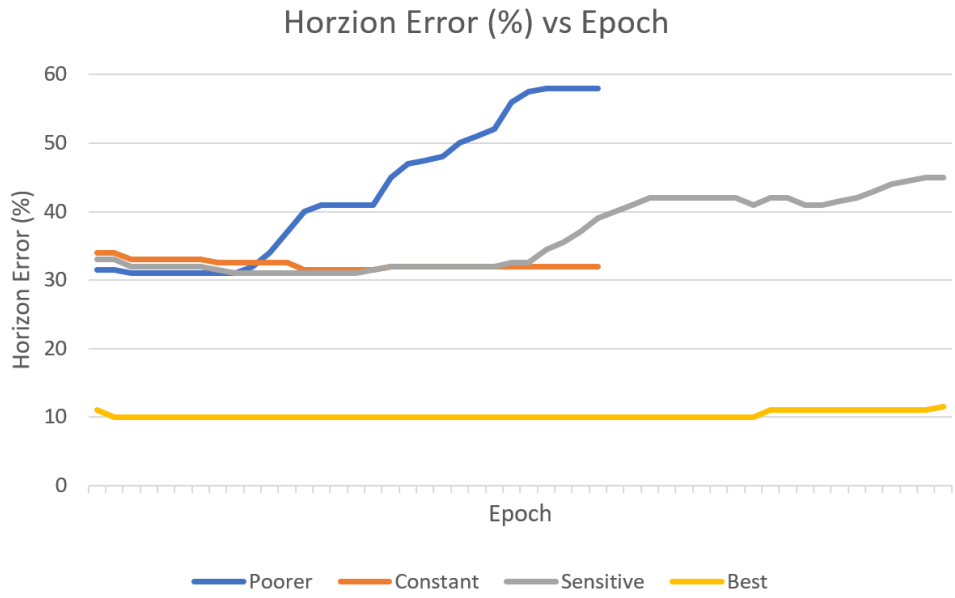


Figure 4.3: Example performance measures of varying configurations of BEAUT

then, with the initial 2015 PIT count, simulated the population for 3 years while adding 82 individuals per month as per the distribution defined before. With these predictions, their WMAEs were also calculated to fairly compare against BEAUT. Consider Table 4.15 as BEAUT's forecast for the 2018 PIT count while noting that the "Hidden Homeless" and "Not Homeless" states are ultimately **not** compared when evaluating. Additionally, in the discussion section, the poorer predictions will be analyzed by looking at how those states transpired in the training dataset to provide further insight.

The performance of the **percentage** distribution will first be presented. Figure 4.4 shows the weighted performance, in the form of a stacked bar, where BEAUT was amongst the majority of the comparison models but fell slightly more towards the best ones. Note that the y-axis represents how many *configurations* for the three comparison models **only** that fell under the covered area of the bar chart on the x-axis, whereas only the best configuration's forecast is shown for BEAUT in the form of a line. Next, the **value** performance of these

	Male	Female		
Street	418	128		
Shelter	367	184		
Hidden Homeless*	-	-	385	144
Not Homeless*	-	-	1437	536
Transitional Housing	962	432		
Institution	461	243		
	2208	**987**		

Table 4.15: BEAUT's forecast for the 2018 PIT count. *Note that hidden homeless and not homeless were only projected as these states were learned from the training data.

models will be presented. Figure 4.5 shows the weighted difference in individuals from the expected output where BEAUT greatly outperforms the top performing comparison model.

4.7 Discussion

When comparing the forecasted 2018 PIT count in Table 4.15 to that of the actual data in Table 4.6, one can see shortcomings in the predictions of (1) institution and (2) males in shelter. For (1), consider the plot in Figure 4.6 as the number of males and females in institution, over time, from the AHCS dataset. It can be seen that BEAUT would be expected to learn that this state mostly increases in population size and from this, it would've applied that to the forecast from 2015 to 2018. For (2), consider another plot from the AHCS dataset in Figure 4.7 that shows the male shelter population over time. It can be seen that BEAUT would've learned an initial increase then a significant **decrease** in the population over time that it would then apply to the 2018 forecast.

When exploring other applications of the proposed "BEAUT" model, a few key-points regarding the specific implementation presented in this chapter need to be taken into account. There are a total of 95,580 data points in the training dataset (AHCS)- that is, every state

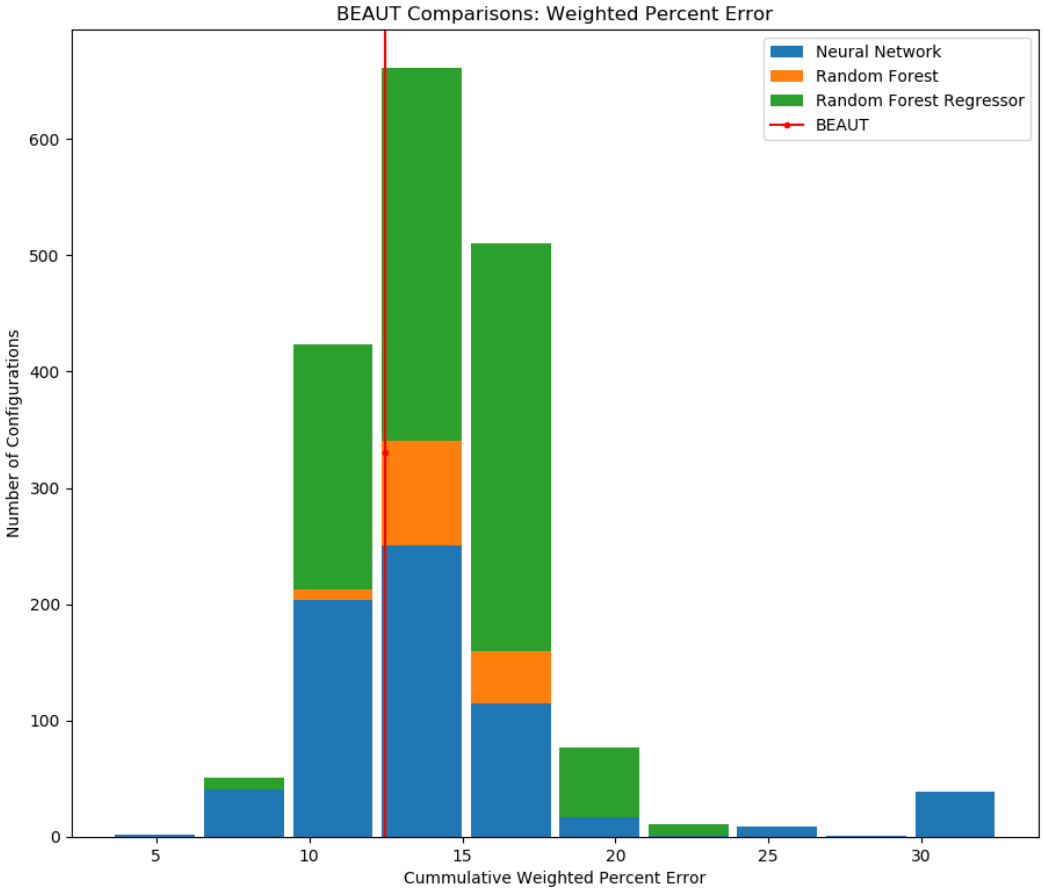


Figure 4.4: The result of the weighted percentage performance metric showing how BEAUT performed in comparison to black box models.

recording, for each individual, for each time period. In the original dataset, researchers on the project looked at the weekly transitions between states and used Equation 3.1 to get the actual probabilities. That approach, however, averaged all of the probabilities to produce a single transition probability matrix whereas our approach only averages for each week of the year (i.e., 52 matrices in total). Therefore, this would be generalizable for any agent-based population performing state transitions as the probabilities are simply the percent

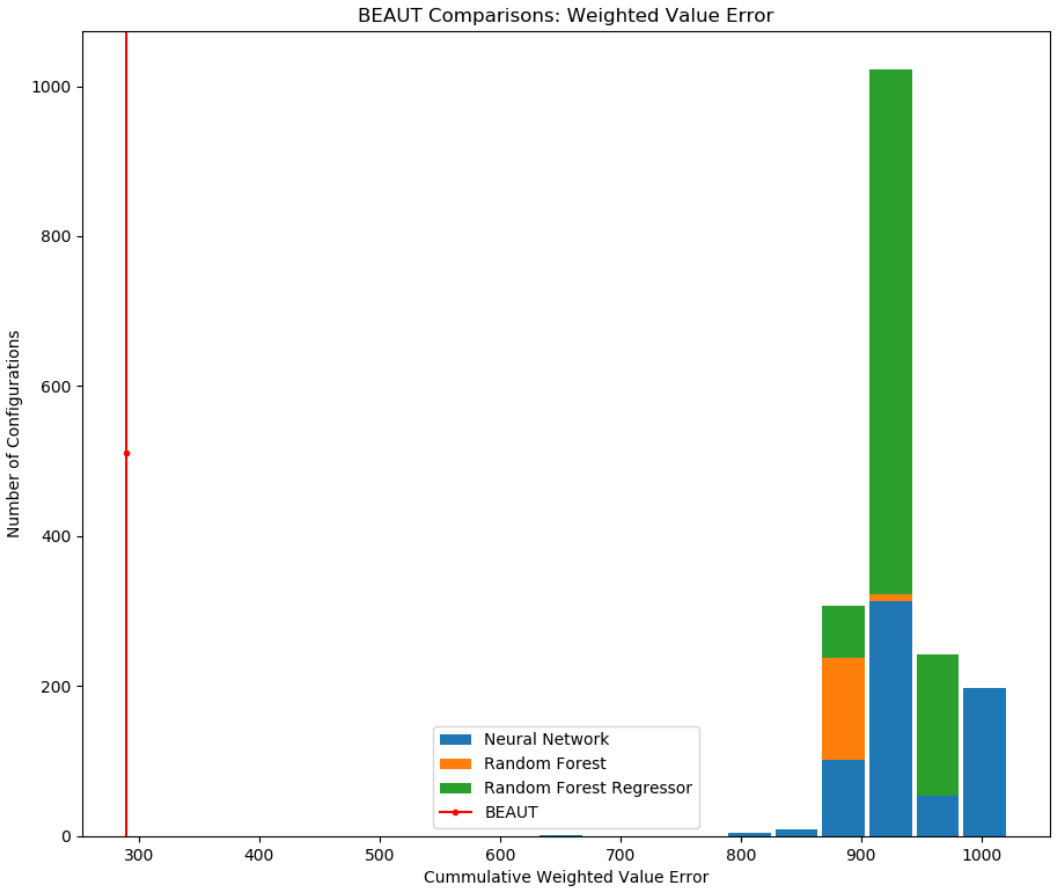


Figure 4.5: The result of the weighted value performance metric showing how BEAUT performed in comparison to black box models.

chance that an agent in a given state will transition to another state at a given time period. An "agent" in this context could be generalized even further by describing it as a dataset of individual transitions over time. However, in this particular application using AHCS, the dataset is from Montreál which means that the trained model is specific to the patterns learned from that and therefore not generalizable; rather, the process to produce this trained model is completely generalizable.

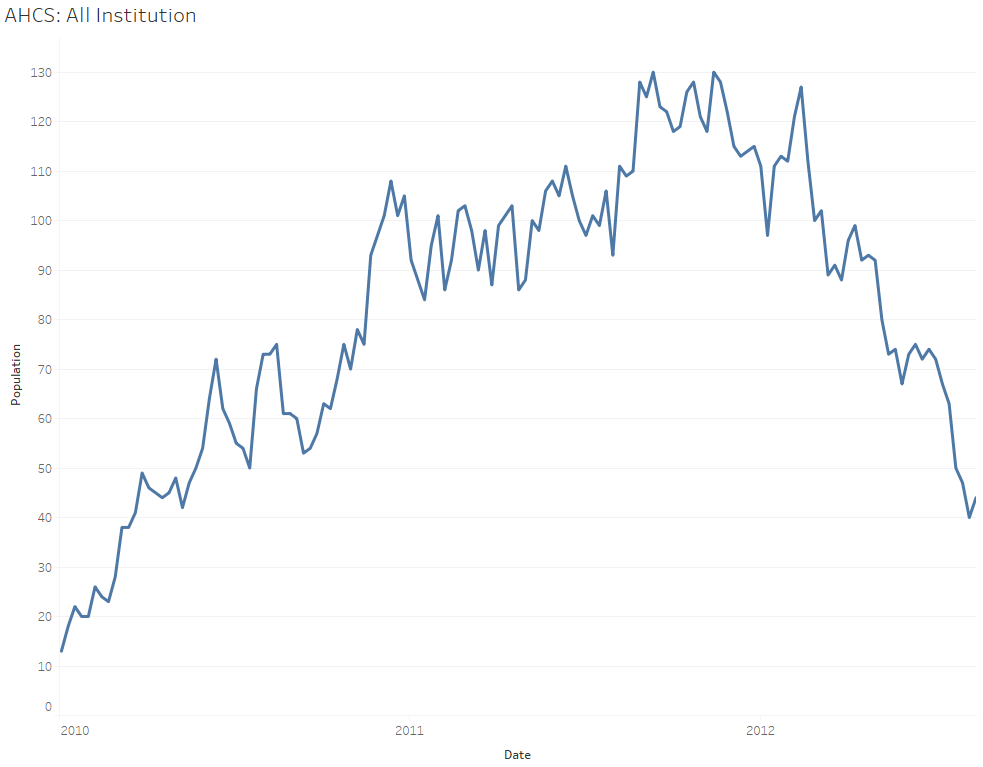


Figure 4.6: The total institute population in the AHCS dataset over time.

4.8 Conclusion

This chapter extended the work shown previously (Chapter 3) to introduce improvements in both the model and dataset. Before, the entire AHCS population was grouped into one cohort which resulted in key information about each individual being lost and the model was unable to account for fluctuations in the population. In addition to this, the previous approach did not demonstrate its ability to forecast future numbers which is important for policy makers to trust its results. The work presented here made use of the limited data available by grouping individuals into 24 sub-populations based on attributes specific to them so that very fine-tuned transition probabilities could be derived. To do this, 24 instances of the previously named MDQL with MNFQ algorithm work indepently on each sub-population

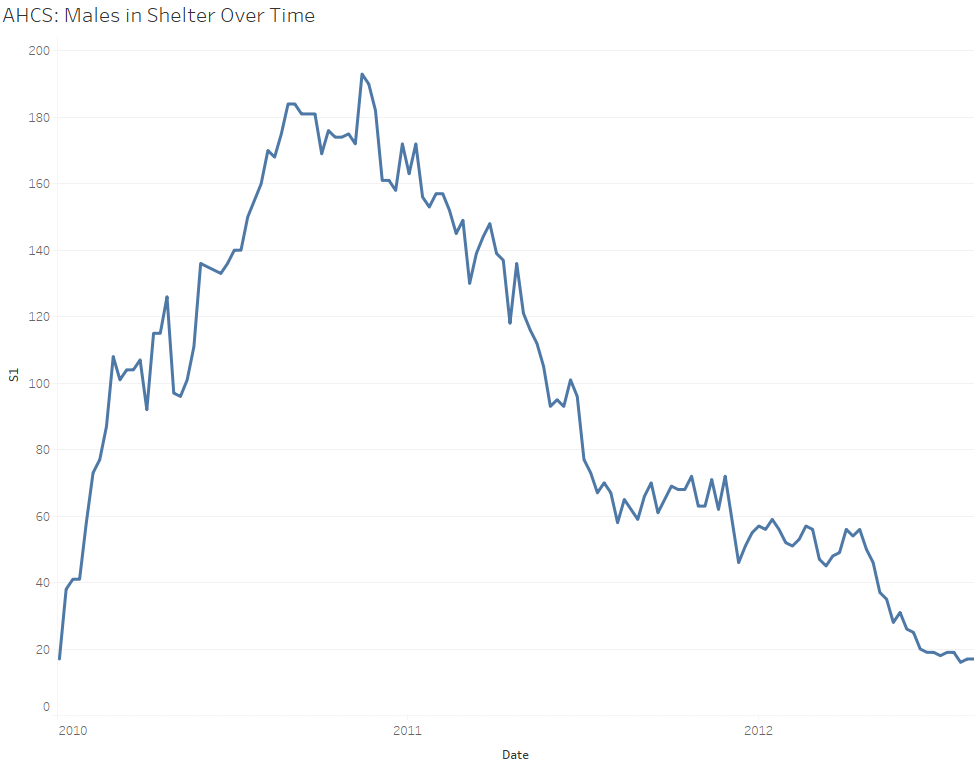


Figure 4.7: The male shelter population in the AHCS dataset over time.

as an overall model named **BEAUT**. As a solution to the fluctuating populations, adaptable parameters were introduced to specifically learn these changes and modify the population over time. These resulted in BEAUT obtaining a $\sim$ 9.8% horizon error, overall, when predicting the last 10 weeks of the training dataset.

With this set of improvements implemented, the model was then demonstrated on PIT counts done in the region, 3 years apart, after the AHCS dataset's timeline to get a true evaluation of its forecasting ability. To get an idea of how its performance fared, 2,040 configurations of 3 different black-box models were evaluated as well. The results showed that BEAUT was amongst the majority of configurations when looking at how the forecasted population was distributed amongst each of the homelessness states with an cummulative weighted error of $\sim$ 12.47%. When looking at the errors for the actual number of individuals

predicted across each state, however, BEAUT greatly outperformed the other models with a cumulative weighted error of ~ 289 individuals. This shows BEAUT's forecasting ability and highlights its importance for policy makers as it also provides them with a set of transition probabilities to fully understand how these predictions were made.

Chapter 5

TentNet: Deep Learning Tent Detection Algorithm Using A Synthetic Training Approach

All of this chapter was published in the following peer-reviewed conference article [34]:
• Fisher, A., Mohammed, E., & Mago, V. (2020). TentNet: Deep Learning Tent Detection Algorithm Using A Synthetic Training Approach. In 2020 IEEE Systems, Man, & Cybernetics (SMC). IEEE.

To continue research efforts on the homelessness domain, I produced this work in a deep learning course that presents a novel approach at using artificial intelligence to detect tents in satellite images with the purpose of contributing to this important social domain. The two main contributions here are as follows: an architecture called "TentNet" that reduces the output of deep convolutional neural networks to a binary output and a data enhancement technique referred to as "synthetic training" that utilizes generative adversarial networks to create a training dataset when the number of real data-points is limited. My role in this publication covered all aspects of the work from research and development to implementation and writeup of the article. We chose IEEE SMC because of its relevance to our work and best student paper award opportunity.

5.1 Introduction

Satellite imagery shows that in some urban parks of Canada, homeless individuals have setup tents to live [44]. Recent attempts have been made by this team [35, 71] to use machine learning and optimization techniques to solve some of the problems faced by homeless individuals. However, there has been no previous study that tried to find the dwelling status of this vulnerable group. One of the main reasons for the lack of such studies is the unavailability of data. As an example, there are approximately 1,600 parks in Toronto, and creating a dataset of these images manually would be a tedious and unreliable process.

Recent tent-detection implementations [131, 117] use mathematical techniques that make a number of assumptions about the characteristics of the tents and only focus on a handful of images. This results in a problem-specific implementation that would have difficulty handling variation in a larger dataset. To resolve this issue, deep learning techniques such as convolutional neural networks (CNNs) could be used as they aim to *learn* the features of these images rather than making assumptions beforehand. The main limitation with this approach, however, is that a smaller dataset results in a smaller number of features available for the algorithm to learn- which is indeed the case with datasets of tents in satellite images. To address this problem, data from an existing project where objects in satellite imagery were classified (such as xView [77]) can first be used to build a small dataset of tents. Then, a generative adversarial network (GAN) can be trained on this dataset with the goal of producing *synthetic* satellite images of tents to increase the overall size of the dataset. The result would provide a significant number of features for the deep learning model to learn.

The objective of the proposed work can be simplified to training a deep learning technique to detect small objects in satellite images. So, these methods will also be evaluated on real

satellite-image-datasets that have been created for detecting ships[1] and planes[2]. This is to provide a proof-of-concept to lead up to the desired problem of detecting tents in satellite images. The motivation for this is to provide more machine learning solutions in the area of homelessness and, if the work were to be implemented in a real life scenario, to help policy makers know the dwelling situation of homeless individuals so they can effectively provide necessary services.

This chapter begins with a background section to describe important aspects (for this work) of both convolutional neural networks and generative adversarial networks. Then, previous implementations to detect tents in satellite images [131, 117] will be discussed– along with their shortcomings– to understand how they approached this problem. Next, the models used in this work will be introduced along with the procedure taken to train and test them. Finally, the results of each model will be shown along with a discussion that explains each outcome.

5.2 Background

Convolutional Neural Network (CNN)

Applying a CNN to this problem will allow for automatic feature extraction and eliminate the necessity of making any assumptions about the features beforehand. The types of layers in these networks that are relevant for this work are as follows: activation function, dense layer, dropout layer, batch normalization, and convolution layer [73]. The activation function is a mathematical definition that describes how a given input will be transformed to the output of that layer. For this work, the authors used three activation functions:

[1] `https://www.kaggle.com/rhammell/ships-in-satellite-imagery`
[2] `https://www.kaggle.com/rhammell/planesnet`

1. ReLU [138]: outputs the maximum of $[0, input]$

2. Softmax [84]: outputs a value between $[0, 1]$

3. LeakyReLU [138]: if $input < 0$ it outputs $\alpha \times input$ where the model uses $\alpha = 0.3$ else it outputs $input$

These functions help reduce the overall linearity as it proceeds into the following layers. Next, the dense layer is a densely-connected neural network that has a defined output size (in this chapter the authors denote this as how many "nodes" it has) while the dropout layer randomly disables a fraction of the nodes (in this chapter the authors denote this as the "percentage") in these layers as a means of preventing over-fitting. Next, batch normalization makes the mean of the inputs as close to zero as possible while maintaining a standard deviation of 1. This is useful in keeping the data propagating through within a small range so that any outlying data, for example, doesn't have a large impact on the network when training. Lastly, the convolution layers have a defined number of filters that, for this work's purposes, are used to process images as they propagate through the network. These filters contain an array of numbers that apply spatial convolutions to the values of each pixel in the image as a means of extracting features from it [73].

Generative Adversarial Network (GAN)

This work will use two different implementations of generative adversarial networks [52] that will be further described in the model section 5.4. The reasoning for this is to see how the outputs differ between a complex (StyleGAN2 [67]) and a simplistic (DCGAN [105]) network. The general setup for a GAN includes two models: (1) a generator for creating synthetic outputs and (2) a discriminator to compare the generator's outputs to the real dataset. By

working together, these two models challenge each other with the goal of generating an image that can't be discriminated from the real ones. Lastly, another layer that is important to understand is a variance of the convolution layer referred to as a *transposed* convolution layer (or "deconvolutional" layer) [52]. For the generator, the convolution layers are these transposed variations that process inputs in a *reverse* way to that of a convolution layer; instead of extracting features from an image, it combines features together to create an image.

5.3 Related Work

To the best of the authors' knowledge, there are two relevant works where satellite data is used for (1) identifying tents from "displaced populations" due to disasters or conflicts [131] and (2) "dwelling detection" in refugee camps [117]. In [131], the main motivation for the work was to provide a means of getting accurate population counts for displaced refugees. Their novel technique used a chain of mathematical morphology operators which the authors state are "useful in detecting objects which have a clear shape, size, and spectral contrast" [131].

In [117], the main motivation for the work was to provide an automated method to detect "different dwelling types in a refugee camp". Their approach involved creating an algorithmic process that first processes the input image(s) with labeled objects in a "semi-automated" manner to determine characteristics about them such as typical brightness, size, and contrast. Then, they used this knowledge with multiple types of image processing approaches such as image segmentation and edge detection to create three distinct methods: (1) optical analysis, (2) synthetic aperture radar (SAR)-based analysis, and (3) feature-based data fusion [117].

In both works, the main shortcomings are that their methods made many assumptions

about the tents themselves, as well as the quality of the images being presented. Both articles described using high resolution satellite imagery which, although ideal, may not always be an achievable assumption and would likely have a great impact on their performance. In this chapter, the models will be shown to achieve accurate results on a low quality, small quantity dataset through the use of deep learning with CNNs and synthetic images from GANs. The models presented don't require any pre-defined assumptions as they learn the features within the dataset that are important for identifying tents in satellite images.

5.4 Methodology

TentNetwork

This chapter uses three deep network CNN models via transfer learning with weights from training on ImageNet [111]. The first model builds on ResNetV2 [55] (an improved version of ResNet [54]) that, while very complex and deep in nature, utilizes skip connections to maintain a high performance while avoiding the vanishing gradient problem. The second model builds on InceptionV3 [122] which, although is another complex model, utilizes feature concatenation to reduce the overall number of parameters required for training to prevent overfitting. Lastly, the third model builds on MobileNetV2 [113] which is much less complex in nature than the previous two models. The reasoning for these selections was to try methods that are very complex (ResNetV2 [55]), complex (InceptionV3 [122]), and less complex (MobileNetV2 [113]) in terms of the networks' architecture. For each model, this work removed the top-most layer, froze the remainder of the parameters from updating, and attached "TentNet" (as described in Figure 5.1) to begin training on the selected datasets.

Since each of the models being used were initially trained on ImageNet, their outputs

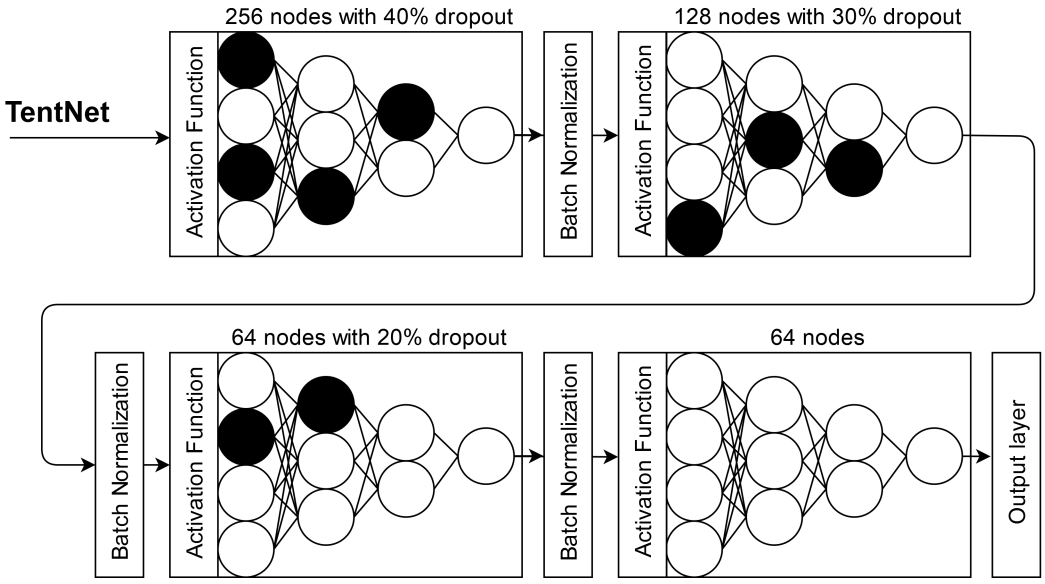


Figure 5.1: An overview of the proposed TentNet architecture.

were designed for 1,000 classes [111]. With each of the datasets used in this chapter, there are only two classes to denote whether or not the desired feature (plane, ship, or tent) is present in the given image. To account for this, the proposed architecture first passes the model's output through an activation function before inputting to a 256 node dense layer. Afterwards, a dropout layer is implemented to reduce the number of nodes used in the dense layer, before passing the output through a batch normalization layer. This process (activation function $\rightarrow$ dense layer $\rightarrow$ dropout layer $\rightarrow$ batch normalization layer) is repeated two more times until the last 64 node layer is reached. Then, the output from this network goes through a batch normalization layer then activation function before finally inputting to the output layer for the final classification. The intuition for this architecture is to slowly reduce the complexity of the CNN model's output to a binary decision, while at the same time trying not to lose any important information along the way.

StyleGAN2

This model was created by NVIDIA labs in late 2019 [67] to further improve the results of the original StyleGAN model [66]. The code-base available to the public[3] includes multiple configurations to run the network in. However, this work will focus on the two main implementations: *config-a* which is the baseline, original model (StyleGAN) and *config-f* which is the improved model (StyleGAN2).

DCGAN

The deep convolutional generative adversarial network (DCGAN) [105] extends the baseline GAN [52] model (as described in section 5.2) with unsupervised learning by using learned features in both the generator and discriminator of the model. The authors modified an architecutre of this from TensorFlow[4] for the purposes of running a grid search (as described in section 5.4) where the goal was to minimize the summation of the generator and discriminator losses; Figure 5.2 shows the architecture used.

The generator first takes a seed as the input to a dense layer which, for this work's purposes, is simply some random data. Then, the outputs of this layer are normalized before being processed by a pre-defined activation function that inputs to a 128-filter convolution layer. The outputs from here are also normalized and ran through the activation function before being input to a 64-filter convolution layer. Lastly, this layer's output is again normalized and processed by the activation function before going to the output convolution layer to generate the resulting image.

The discriminator first takes an image as the input to a 64-filter convolution layer. Then, the output from this is processed by the activation function before having some of the results

[3] `https://github.com/NVlabs/stylegan2`
[4] `https://github.com/tensorflow/docs`

dropped by a dropout layer. Next, the values left are input to a 128-filter convolution layer then processed by the activation function again. Lastly, these values go through another dropout layer before inputting to the output dense layer that provides a probability as to whether or not this is a real image.

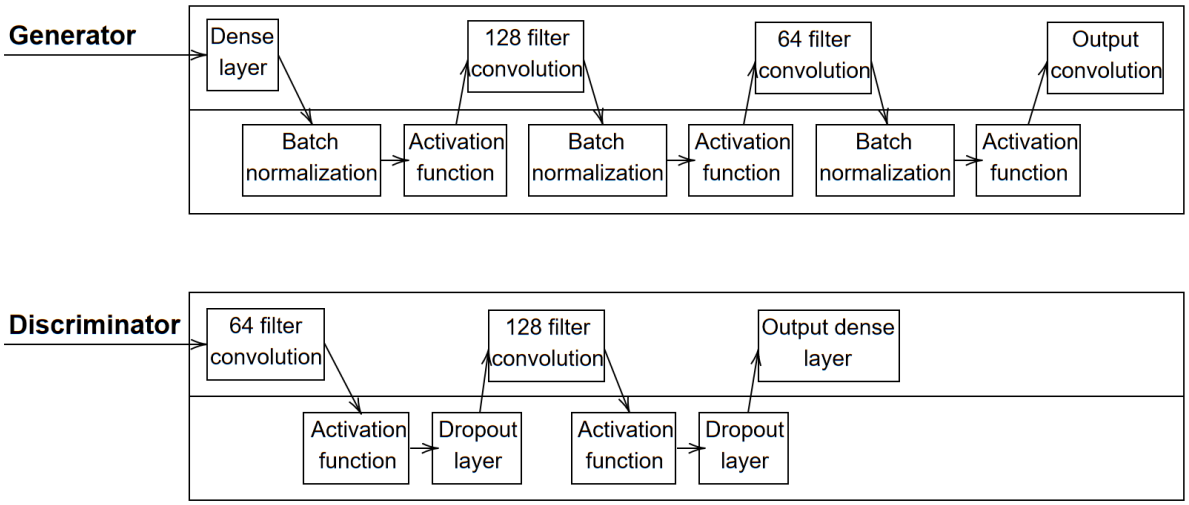


Figure 5.2: An overview of the layout that the DCGAN in this chapter uses.

Dataset Processing

Planes and Ships

These are two datasets that are publicly available under the CC BY-SA 4.0 license on `Kaggle.com`. The author of both used data from `Planet.com` imagery to create 32,000 20px×20px RGB images for the planes dataset and 4,000 80px×80px RGB images for the ships. The 32,000 in the planes dataset consists of 8,000 positive images (contains a plane) and 24,000 negative images (does not contain a plane), while the 4,000 in the ships dataset

consists of 1,000 positive images and 3,000 negative images. Once the datasets were retrieved, each image was converted to a numeric array (three dimensional for RGB), along with their label of being a positive or negative image.

xView Dataset

To achieve the end goal of producing an accurate means of detecting tents in satellite imagery, the authors used a subset of data from the xView dataset [77]. This project classified millions of features in thousands of satellite images with one of the categories being "Hut/Tent". This entire dataset isn't openly available to the public but a subset is that consists of ∼18.2GB of images. After parsing this data, the authors found that there were 50 images in the subset containing a total of 245 features classified as "Hut/Tent". The first step was to analyze these features and determine their dimensions as individual images to create a dataset for these models. The authors found that the largest width in this feature set was 260px while the largest height was 308px, but that the averages were only 32px and 35px respectively. Therefore, the authors decided on choosing 64px×64px as the maximum size for the real-tents-dataset which produced 218 "Hut/Tent" images that met this constraint- or roughly 89% of the images available to us. Figure 5.3 shows three images from this resulting dataset.

CNN Model Fine-Tuning

For each of the three models described in section 5.4, a grid search was ran on the attached TentNet architecture that differed in both the optimization method and activation layers to find an optimal configuration. Namely, this search used the activation methods ReLU [138] and Softmax [84], along with optimization methods Adagrad [72] and Adam [72] where each combination ($2 \times 2 = 4$) was 3 cross-fold-validated for a total of 12 fits on all datasets for

Figure 5.3: Three examples of "Hut/Tent" images that the authors extracted from the xView dataset [77].

each model. Using the best parameters from this process, on the planes and ships datasets, 10 cross-fold-validation was ran on the configuration to produce an accurate performance measure to compare the three models. When running the cross fold validation, the authors used a stratified fold to ensure the class distribution (positive and negative) was as even as possible and, additionally, provided the model with a weight for each class that was based on the number of images present for it in the dataset where a high weight meant that the class was more important to learn (i.e., it was under-represented in the class distribution). This was particularly important for the planes and ships datasets as the number of images with the desired feature in it couldn't be assumed to be balanced with that of images without the feature.

As for the tents dataset, since the authors only had 218 real images of "Huts/Tents", this work kept them exclusively for the positive (i.e., tent) images in the **testing** dataset along with 200 randomly selected features in the xView dataset [77] classified as "Shed" or "Building" for the negative (i.e., no tent) images. Then, the authors used 1,000 generated positive images (process described in section 5.4) along with 1,000 "Shed" or "Building" images for the **training** dataset. The evaluation on this dataset was simply based on the

testing accuracy achieved by each model.

The best parameter configuration from the grid search for each model is shown in Table 5.1 for the planes and ships datasets. For the tents dataset, these configurations are shown in Table 5.2 for both StyleGAN2 and DCGAN when they were respectively used to create the positive class synthetic images (i.e., satellite images with tents). In all datasets, the model trained on very small batch sizes of 8 images which is important to consider when working with the small dataset of tents.

Model	**Planes Activation**	**Ships Activation**	**Planes Optimization**	**Ships Optimization**
ResNetV2	ReLU	Softmax	Adam	Adam
InceptionV3	Softmax	Softmax	Adagrad	Adam
MobileNetV2	ReLU	Softmax	Adam	Adam

Table 5.1: The best configurations for each model on the planes and ships datasets.

Model	**StyleGAN2 Activation**	**DCGAN Activation**	**StyleGAN2 Optimization**	**DCGAN Optimization**
ResNetV2	ReLU	Softmax	Adagrad	Adagrad
InceptionV3	ReLU	ReLU	Adagrad	Adagrad
MobileNetV2	Softmax	ReLU	Adagrad	Adam

Table 5.2: The best configurations for each model on the tents datasets with synthetic images from StyleGAN2 and DCGAN.

GAN Model Design and Fine-Tuning

StyleGAN2

When the authors used StyleGAN2, both config-a and *config-f* were ran on the real dataset of "Hut/Tent" images as described in section 5.4. The training method in this code-base provides an option called "mirror augmentation" that was used, which simply rotates the

input images to produce a larger dataset for the model to learn. The output of these models as shown in Figure 5.4 appear to be abstract and indistinguishable. The authors attribute this to the small resolution of the input images as well as the overall small size of the dataset and the fact that StyleGAN is quite complex in architecture. However, the authors noted that config-f produced the *more* distinguishable output so an analysis of how the features present in these generated images impact the models when training will be shown.

(a) Output of config-a from StyleGAN2. (b) Output of config-f from StyleGAN2.

Figure 5.4: An example output from *config-a* (a) and *config-f* (b) of StyleGAN2 [67].

DCGAN

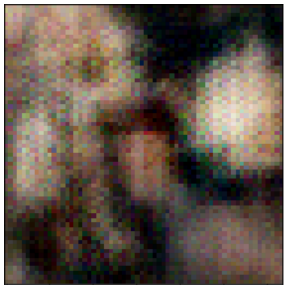
To create an optimal configuration, a grid search was ran on an implementation of this model that was modified (as described in section 5.4). The parameters that differed in both the generator and discriminator include activation functions Softmax [84] and LeakyReLU [138], as well as optimization functions Adagrad [72] and Adam [72]. A description of how these models were initialized can be found in Algorithm 3 for the generator and Algorithm 4 for the discriminator. For the generator only, the authors also differed whether or not biases were used in the convolutional layers as these act as an offset to the outputs of each node in the network. For the discriminator only, the authors also changed the percentage of the

dropout layers to either 0.3 or 0.4 as the default value in the original impelmentation was 0.3.

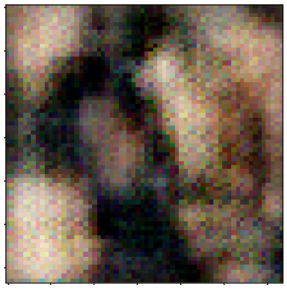
For each possible combination of the generator parameters, they were paired with each of the possible combination of discriminator parameters for a total of 64 DCGAN models to run a grid search on. The result of this is shown in Table 5.3, along with an example output in Figure 5.5. A visual comparison between the outputs from DCGAN (Figure 5.5) and StyleGAN2 (Figure 5.4) shows that DCGAN produces the more "realistic" results in this context when compared to the actual dataset as shown in Figure 5.3. The authors attribute this observation to the fact that DCGAN has a much more simplistic architecture than that of StyleGAN2 and as a result, is able to process the small dataset with a better output.

Parameter	**Generator**	**Discriminator**
Activation	Leaky ReLU	Leaky ReLU
Optimization	Adagrad	Adam
Use Bias	True	N/A
Dropout	N/A	0.4

Table 5.3: The best configurations for the DCGAN [105] generator and discriminator on the tents dataset.



(a) Example 1 from DCGAN.



(b) Example 2 from DCGAN.

Figure 5.5: Example outputs from the optimized DCGAN model [105].

5.5 Results

Planes Dataset

With all three of the pre-trained models, the minimum input size allowed was 75px×75px. Since the planes dataset contains images of size 20px×20px, this work simply added padding around the images to reach this requirement. As the results will show, compared to the ships dataset, this likely had an impact on the performance for the planes dataset but, nevertheless, provides a useful observation to show how impactful the pre-processing steps are when evaluating these models. Using the best configurations as shown in Table 5.1, the authors evaluated TentNet on ResNetV2 (using the "ReLU" activation function and "Adam" optimization function), InceptionV3 (using the "Softmax" activation function and "Adagrad" optimization function), and MobileNetV2 (using the "ReLU" activation function and "Adam" optimization function). The results were retrieved after ten epochs across each of the ten folds created in the dataset as shown in Figure 5.6. An observation can be made that, for MobileNetV2's results, it appears to be over-fitting on the 5th fold despite the author's efforts to balance the datasets and provide weights for the classes when training. Since this dataset is entirely composed of real images, one would expect the training and testing accuracy to be close to one another to show that there isn't over/under-fitting. Additionally, a high testing accuracy would show that the model performs well on this dataset while taking into consideration that the images needed to be padded.

A tabulated version of these of the results is shown in Table 5.4 which include the average training and testing accuracy across the 10 folds for each model. As this chapter has described before and will see when analyzing the ships dataset results, the padding of these images appears to have caused a hit in the accuracy of these models. Nevertheless, in terms of

Figure 5.6: The results of the three models with the proposed TentNet architecture across 10 folds of the planes dataset.

training and testing accuracies, ResNetV2 performed the best with a training accuracy of 88.68% and a testing accuracy of 84.44%.

Model	Training	Testing
ResNetV2	88.68%	84.44%
InceptionV3	84.77%	75.00%
MobileNetV2	91.41%	78.41%

Table 5.4: The average training and testing accuracy for each model across the 10 folds of the planes dataset.

Ships

Using the configurations shown in Table 5.1 (all three models used the "Softmax" activation function and "Adam" optimization function), the authors retrieved the results of ResNetV2, InceptionV3, and MobileNetV2 after 10 epochs across each of the ten folds created in the dataset. These are shown in Figure 5.7. In MobileNetV2's results, another instance of over-fitting can be observed in the 1st fold. However, the model appears to level out for the remainder of the evaluation and provides a good performance in terms of not over/under-fitting the data. This improvement for MobileNetV2 over the planes dataset's results could be attributed to not needing to pad the images.

The tabulated version of the results for this dataset are shown Table 5.5 which include the average training and testing accuracy across the 10 folds for each model. The performance on this dataset shows improvement over that of the planes dataset's results for **all** models. However, MobileNetV2 was performed the best with a training accuracy of 96.91% and a testing accuracy of 94.90%.

Tents

As with the plane's dataset images, this work needed to pad the tent's dataset images to meet the 75px×75px minimum requirement. However, since these images were already 64px×64px,

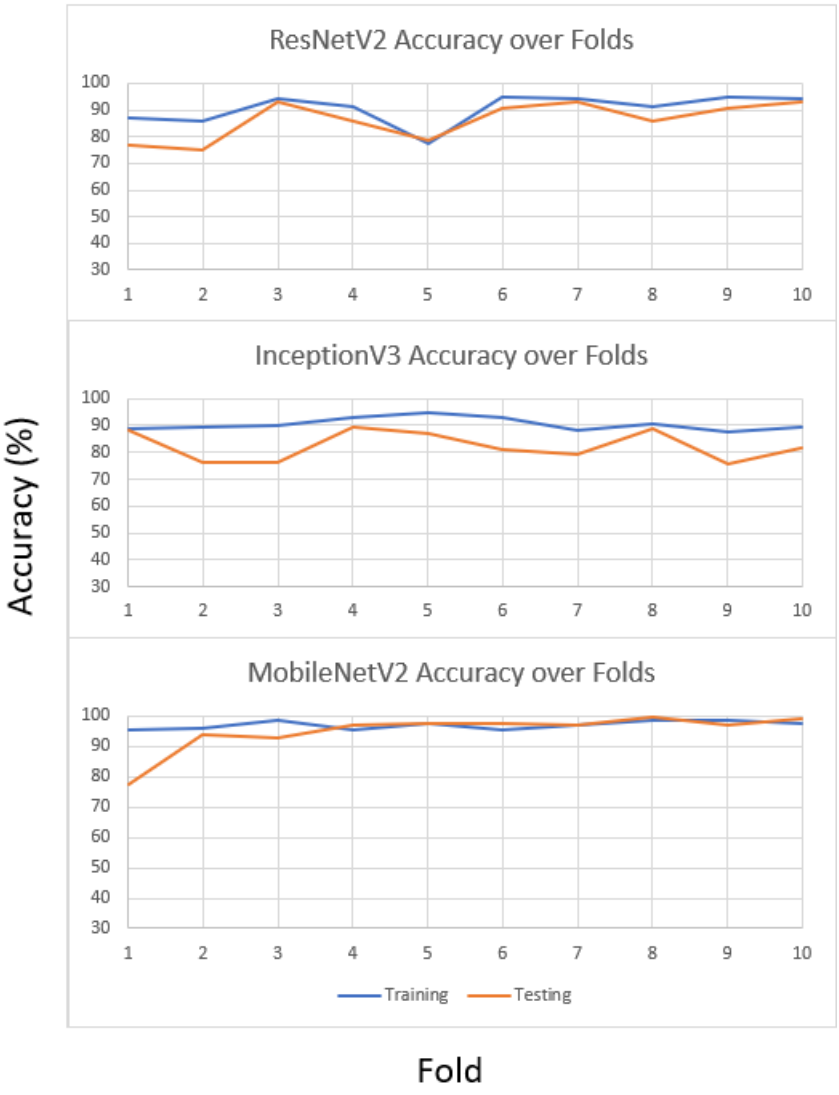


Figure 5.7: The results of the three models with the proposed TentNet architecture across 10 folds of the ships dataset.

the padding was minimal in comparison to that of the planes. To summarize the results of the models for both the StyleGAN2 and DCGAN datasets, consider Table 5.6. When comparing the two, one can see that using the DCGAN's outputs for training on the synthetic images resulted in higher training accuracies for all models. This could be attributed to the visual

Model	**Training**	**Testing**
ResNetV2	90.50%	86.30%
InceptionV3	90.36%	82.34%
MobileNetV2	96.91%	94.90%

Table 5.5: The average training and testing accuracy for each model across the 10 folds of the ships dataset.

comparison of the outputs from StyleGAN2 in Figure 5.4 and DCGAN in Figure 5.5 to that of the real dataset as shown in Figure 5.3.

Model	**StyleGAN2 Synthetic Training**	**DCGAN Synthetic Training**	**StyleGAN2 Real Testing**	**DCGAN Real Testing**
ResNetV2	89.65%	95.85%	47.85%	73.68%
InceptionV3	96.50%	81.75%	35.17%	48.80%
MobileNetV2	99.45%	98.55%	47.85%	47.85%

Table 5.6: The results of these models on the tent dataset with training accuracies on synthetic images from StyleGAN2 running the *config-f* configuration as well as from the optimized DCGAN, then testing accuracies on the real images.

Looking at the results in Table 5.6, it may appear that the data is being over-fitted. However, the layout of the dataset used needs to be taken into consideration. Each model learned the positive (tent) feature strictly from synthetic images which resulted in a higher training accuracy. Then, when testing, the positive class was represented using **only** real images to provide a fair evaluation of their performance from the synthetic training. The best performing model was ResNetV2 which demonstrated a testing accuracy of 73.68% when training on synthetic images from the DCGAN. An example of how the process for this particular configuration was performed can be observed in Figure 5.8 but please note that the synthetic dataset was only generated via DCGAN **once** then used for InceptionV3 and MobileNetV2 also; the figure shows this process simply to provide more clarification.

The proposed approach helps overcome the challenge of having a small dataset by generating a large synthetic dataset that contains features learned from the originals using a GAN. This is then used along with a sample of non-tent images to train TentNet so that the parameters within the model can be adjusted accordingly when classifying whether or not a tent is present. Finally, the model is then evaluated on the small, real dataset to see how well it learned the features that the GAN originally learned.

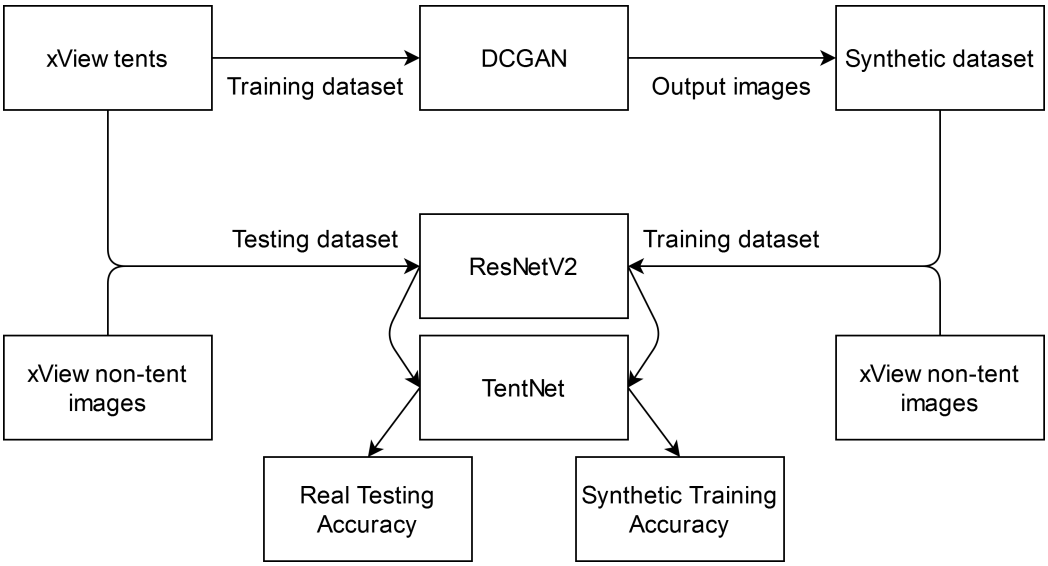


Figure 5.8: The process performed to evaluate the accuracy of TentNet on the tents dataset.

5.6 Discussion and Conclusion

This chapter presented three CNN models that used transfer learning from ResNetV2, InceptionV3, and MobileNetV2 with the goal of identifying whether or not a tent is present in satellite imagery. As a baseline performance measure, these models were evaluated on two datasets made for detecting (1) planes and (2) ships within satellite imagery. For the planes dataset, the ResNetV2 model performed the best with a average testing accuracy,

over 10 folds, of 84.44%. For the ships dataset, the MobileNetV2 model performed the best with an average testing accuracy, over 10 folds, of 94.90%. When comparing the results between these two datasets, it should be noted that each model required a minimum input size of 75px×75px. The planes dataset didn't meet this requirement (20px×20px) so they were simply padded to demonstrate how much the pre-processing steps impact these models' performances.

For the tents aspect, since it is a very unique feature, the authors needed to come up with a solution to provide a sufficient amount of data to these deep learning models. By utilizing the xView dataset [77], this work created the testing dataset with 218 64px×64px RGB images of features classified as "Hut/Tent" for the positive class images, and 200 "Building" or "Shed" features from this dataset for the negative class images. For the training dataset, the authors presented two GAN implementations that were used to create 1,000 synthetic positive images, and retrieved 1,000 "Building" or "Shed" features from the xView dataset for the negative images. Similar to the planes dataset, these images did need to be padded as well to meet the 75px×75px requirement.

When using the StyleGAN2 implementation to train the best performing model on this dataset, ResNetV2 achieved a testing accuracy of 47.85%. This could be attributed to the abstract synthetic image outputs that were produced as seen in Figure 5.4. When using DCGAN, however, ResNetV2 was the best performing model again but this time with an accuracy of 73.68%. This improvement can be attributed to the output quality from the simplistic architecture in the DCGAN being much more distinguishable than that of the output from StyleGAN2's much more complex architecture when visually compared to the very small real dataset.

With the methods provided in this chapter, the authors have improved the approaches previously defined for tent-detection algorithms. In other works, many assumptions needed

to be made and defined about the characteristics of the tents before it could process the image. Using deep learning, the authors eliminated this requirement as these models instead learn the features of the images through training on a given dataset. Additionally, in previous works the models processed only a handful of high resolution images while the proposed approach uses synthetic images from a GAN to learn, then approximately 218 real images to test from the xView dataset [77].

5.7 Future Work

An interesting application of this work could be to determine *how many* tents are in the satellite image rather than *whether or not there is one*. Some modifications to the models presented in this chapter would be needed such as region-of-interest (RoI) pooling [21] to adapt the output from strictly being a positive/negative classification. Based on the uniqueness of this problem and the assumptions made in current implementations [131, 117] designed for counting tents in a satellite image, this could certainly be a beneficial extension. Additionally, the use of other images sources– such as ones from drones– could be explored as an alternative to satellite images but the problem arises as to whether or not a large enough dataset exists with such features. To the best of the authors' knowledge, this is not the case but the synthetic approach used in this chapter could be a possible solution.

Another area of this work that could be extended would be running larger grid searches to explore more activation and optimization functions with these models, as well as different parameters such as kernel sizes and the number of nodes. Since the architectures this work transferred learning from were very complex and large in nature, the computational overhead required was immense- for example, a single fold in Figure 5.6 took upwards of 5 hours in some instances. Similarly, training the discriminator in the DCGAN with all datasets (planes,

ships, and tents) would also be an interesting extension of this work. This would provide more satellite-imagery-features for it to learn and, in turn, provide a greater challenge to the generator when trying to produce realistic tent outputs.

Chapter 6

Conclusion

In this thesis, a number of solutions to social and health issues were presented that use simulation and machine learning approaches. The first theme was based on an agent-based model using deep learning to simulate homelessness populations as presented in Chapters 3 and 4. This was done by modifying two pre-existing algorithms– deep q-learning and neural fitted q-iteration– to work together to produce an explainable artificial intelligence through the use of transition probability matrices. This is important to researchers in this social domain as they can easily understand how the model is simulating these populations with this crucial piece of information. As a result, these chapters provided a way of testing different approaches to reducing homelessness without a real world implementation.

In the last chapter of this thesis (Chapter 5), a transfer learning architecture was presented to detect tents in satellite images for applications such as determining where homeless individuals may be living. As such a dataset does not exist, a method was also shown that involved using generative adversarial networks to generate a set of synthetic images to be used to train the model. This resulted in the best performing model obtaining a 73.68% accuracy on real images after only being trained on synthetic ones.

The work presented here was motivated by the desire to encourage more collaboration to be done between computer and social sciences. This is an important connection to make as simulation and machine learning methods have the ability to process information in greater quantities and speeds, while still learning more about the patterns of such data than we can easily comprehend. The issues posed throughout each chapter can affect each and every one of us so having a solution to alleviate or resolve those problems is of great benefit.

Bibliography

[1] "An empirical comparison of machine learning models for time series forecasting". In: *Econometric Reviews* 29.5 (2010), pp. 594–621. ISSN: 07474938. DOI: 10.1080/07474938.2010.481556.

[2] William Ascher. *Forecasting: An Appraisal for Policy-Makers and Planners*. Baltimore: Johns Hopkins University Press, 1980. DOI: 10.2307/2327228.

[3] Aubry T, Goering P, Veldhuizen S, Adair C, Bourque J, Distasio J, et al. "A Randomized Controlled Trial in Five Canadian Cities of the Effectiveness of Housing First with Assertive Community Treatment for Persons with Serious Mental Illness and a History of Homelessness". In: *Psychiatric Services* 67(3) (2016), pp. 275–81.

[4] J Badham. *A compendium of modelling techniques*. Project Report. Canberra, Australia: The Australian National University, May 2010. URL: http://epubs.surrey.ac.uk/771881/.

[5] Osman Balci. "Validation, verification, and testing techniques throughout the life cycle of a simulation study". In: *Annals of Operations Research* 53.1 (1994), pp. 121–173. DOI: 10.1007/bf02136828.

[6] Batata, O., Augusto, V., Xie, X. "Mixed machine learning and agent-based simulation for respite care evaluation". In: *Winter Simulation Conference* (2018), pp. 2668–2679.

[7] Dimitris Bertsimas and Nathan Kallus. "From Predictive to Prescriptive Analytics". In: *Management Science* February 2020 (2019). ISSN: 0025-1909. DOI: 10.1287/mnsc.2018.3253. arXiv: 1402.5481.

[8] Simon Elias Bibri. *The anatomy of the data-driven smart sustainable city: instrumentation, datafication, computerization and related applications*. Vol. 6. 1. Springer International Publishing, 2019. ISBN: 4053701902. DOI: 10.1186/s40537-019-0221-4. URL: https://doi.org/10.1186/s40537-019-0221-4.

[9] Louis G Birta and Gilbert Arbez. *Modelling and Simulation: Exploring Dynamic System Behaviour*. Springer Nature, 2013, pp. 19–51.

[10] Mohamed A Bouhlel and Joaquim RRA Martins. "Gradient-enhanced kriging for high-dimensional problems". In: *Engineering with Computers* 35.1 (2019), pp. 157–173.

[11] Mohamed A Bouhlel et al. "A Python surrogate modeling framework with derivatives". In: *Advances in Engineering Software* (2019), p. 102662.

[12] Mohamed A Bouhlel et al. "An improved approach for estimating the hyperparameters of the kriging model for high-dimensional problems through the partial least squares method". In: *Mathematical Problems in Engineering* 2016 (2016).

[13] J B Bricker and S J Tollison. "Comparison of Motivational Interviewing with Acceptance and Commitment Therapy: A conceptual and clinical review". In: *Behavioural and cognitive psychotherapy* 39.5 (2011), pp. 541–559. ISSN: 15378276. DOI: 10.1126/scisignal.2001449.Engineering.

[14] Muffy Calder et al. "Computational modelling for decision-making: where, why, what, who and how". In: *Royal Society open science* 5.6 (2018), p. 172096.

[15] Robert J Calsyn et al. "Reliability and validity of self-report data of homeless mentally ill individuals". In: *Evaluation and Program Planning1* 20.1 (1997), pp. 47–54.

[16] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2011), p. 27.

[17] Peter Checkland. *Systems Thinking, Systems Practice*. Wiley, 1981.

[18] Laurens Cherchye, Bram De Rock, and Frederic Vermeulen. "Economic well-being and poverty among the elderly: An analysis based on a collective consumption model". In: *European Economic Review* 56.6 (Aug. 2012), pp. 985–1000. ISSN: 00142921. DOI: 10.1016/j.euroecorev.2012.05.006.

[19] City of Toronto. *Street Needs Assessment Results 2018*. 2018.

[20] R. G. Coyle. *System Dynamics Modelling: A Practical Approach*. Chapman & Hall, 1996.

[21] Jifeng Dai et al. "R-fcn: Object detection via region-based fully convolutional networks". In: *Advances in neural information processing systems*. 2016, pp. 379–387.

[22] Jennifer Davis-Berman. "Older men in the homeless shelter: in-depth conversations lead to practice implications." In: *Journal of gerontological social work* 54.5 (July 2011), pp. 456–74. ISSN: 1540-4048. DOI: 10.1080/01634372.2011.570863.

[23] *Dénombrement des personnes en situation d'itinérance sur l'île de Montréal le 24 avril 2018. Montréal*. 2019.

[24] "Designing fair, efficient, and interpretable policies for prioritizing homeless youth for housing resources". In: *International Conference on the Integration of Constraint*

Programming, Artificial Intelligence, and Operations Research. Springer Cham, 2018, pp. 35–51. ISBN: 9783319930305. DOI: 10.1007/978-3-319-93031-2_3.

[25] Virginia Dignum. "Assessing organisational design". In: *Simulating Social Complexity*. Springer, 2013, pp. 541–562.

[26] Daizong Ding et al. "Modeling extreme events in time series prediction". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 1 (2019), pp. 1114–1122. DOI: 10.1145/3292500.3330896.

[27] Duffin E. *Estimated number of homeless people in the United States from 2007 to 2019 Statista*. 2020.

[28] Eden, T., Knittel, A., & Van Uffelen, R. *Reinforcement Learning*. https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html. Retrieved May 15, 2019.

[29] Joshua M. Epstein. "Why Model?" In: *Journal of Artificial Societies and Social Simulation* 11.4 (2008), p. 12. ISSN: 1460-7425. URL: http://jasss.soc.surrey.ac.uk/11/4/12.html.

[30] Karin M. Eyrich-Garg and Shadiya L. Moss. "How Feasible is Multiple Time Point Web-Based Data Collection with Individuals Experiencing Street Homelessness?" In: *Journal of Urban Health* 94.1 (2017), pp. 64–74. ISSN: 14682869. DOI: 10.1007/s11524-016-0109-y.

[31] Pierre F-TFA. "Europe and Homelessness: Alarming Trends". In: *Second Overview of Housing Exclusion in Europe* 1 (2017).

[32] Christina D. Falci et al. "Predictors of Change in Self-Reported Social Networks Among Homeless Young People". In: *Journal of Research on Adolescence* 21.4 (2011), pp. 827–841. ISSN: 10508392. DOI: 10.1111/j.1532-7795.2011.00741.x.

[33] Seena Fazel et al. "The prevalence of mental disorders among the homeless in western countries: systematic review and meta-regression analysis". In: *PLoS medicine* 5.12 (2008).

[34] A. Fisher, E. A. Mohammed, and V. Mago. "TentNet: Deep Learning Tent Detection Algorithm Using A Synthetic Training Approach". In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2020, pp. 860–867. DOI: 10.1109/SMC42975.2020.9283377.

[35] Andrew Fisher, Vijay Mago, and Eric Latimer. "Simulating the Evolution of Homeless Populations in Canada Using Modified Deep Q-Learning (MDQL) and Modified Neural Fitted Q-Iteration (MNFQ) Algorithms". In: *IEEE Access* 8 (2020), pp. 92954–92968.

[36] Fisher A, Mago V, Latimer E. *Homelessness Simulation Using Modified Deep Q-Learning Algorithms*. URL: https://www.researchgate.net/publication/334113141_Homelessness_Simulation_Using_Modified_Deep_Q-Learning_Algorithms.

[37] Cheryl Forchuk et al. "Retaining a sample of homeless youth". In: *Journal of the Canadian Academy of Child and Adolescent Psychiatry* 27.3 (2018), pp. 167–174. ISSN: 17198429.

[38] Fowler AJ, Hovmand PS, Marcal KE, Das S. "Solving Homelessness from a Complex Systems Perspective: Insights for Prevention Responses". In: *Annual Review of Public Health* 40 (2019), pp. 465–86.

[39] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2001.

[40] S Gaetz, E Dej, and T Richter. *The state of homelessness in Canada 2016*. Canadian Homelessness Research Network, 2016.

[41] Stephen Gaetz and Erin Dej. *A new direction: A framework for homelessness prevention*. Canadian Observatory on Homelessness Press Toronto, Ontario, Canada, 2017.

[42] Bart Gajderowicz. "Artificial Intelligence Planning techniques for emulating agents with application in Social Services". PhD. University of Toronto, 2019.

[43] Bart Gajderowicz, Mark S Fox, and Michael Grüninger. "Requirements for an ontological foundation for modelling social service chains". In: *Proceedings of the 2014 Industrial and Systems Engineering Research Conference*. Ed. by Y Guan and J Liao. Montréal, QC, 2014.

[44] Leanna Garfield. *Before-and-after Photos Show How a Major City's Homelessness Crisis Can Spiral out of Control*. 2018. URL: `http://www.businessinsider.com.au/homelessness-crisis-vancouver-cities-google-street-view-photos-2018-7`.

[45] Philippe Giabbanelli and Rik Crutzen. "An agent-based social network model of binge drinking among Dutch adults". In: *Journal of Artificial Societies and Social Simulation* 16.2 (2013), p. 10.

[46] Philippe J Giabbanelli et al. "Modeling the influence of social networks and environment on energy balance and obesity". In: *Journal of Computational Science* 3.1-2 (2012), pp. 17–27.

[47] GN Gilbert and L Hamill. "Social circles: A simple structure for agent-based social network models". In: *Journal of Artificial Societies and Social Simulation* 12.2 (2009).

[48] Chris Glynn and Alexander Casey. *Homelessness Rises Faster Where Rent Exceeds a Third of Income*. 2018.

[49] Chris Glynn and Emily B. Fox. "Dynamics of homelessness in urban America". In: *Annals of Applied Statistics* 13.1 (2019), pp. 573–605. ISSN: 19417330. DOI: 10.1214/18-AOAS1200. arXiv: 1707.09380.

[50] Paula N Goering et al. "The At Home/Chez Soi trial protocol: a pragmatic, multi-site, randomised controlled trial of a Housing First intervention for homeless individuals with mental illness in five Canadian cities". In: *BMJ Open* 1.2 (2011), pp. 1–18. ISSN: 2044-6055. DOI: 10.1136/bmjopen-2011-000323. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3221290/pdf/bmjopen-2011-000323.pdf.

[51] Goering PN, Streiner DL, Adair C, Aubry T, Barker J, Distasio J, et al. "The At Home/Chez soi trial protocol: a pragmatic, multi-site, randomised controlled trial of a Housing First intervention for homeless individuals with mental illness in five Canadian cities". In: *British Medical Journal* (2011), pp. 1–18.

[52] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[53] Simon S. Haykin. *Neural networks and learning machines*. Prentice Hall, 2009.

[54] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385* (2015).

[55] Kaiming He et al. "Identity Mappings in Deep Residual Networks". In: *arXiv preprint arXiv:1603.05027* (2016).

[56] Tin Kam Ho. "Random decision forests". In: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282.

[57] Sepp Hochreiter and J Urgen Schmidhuber. "Long Shortterm Memory". In: *Neural Computation* 9.8 (1997), p. 17351780.

[58] Homelessness Analytics Initiative. *Access critical national, state, and local information about homelessness and related factors*. 2020.

[59] Richard L Hough et al. "Recruitment and Retention of Homeless Mentally Ill Participants in Research". In: *Journal of Consulting and Clinical Psychology* 64.5 (1996), pp. 881–891.

[60] John T Hwang and Joaquim RRA Martins. "A fast-prediction surrogate model for large datasets". In: *Aerospace Science and Technology* 75 (2018), pp. 74–87.

[61] Stephen W Hwang. "Homelessness and health". In: *Cmaj* 164.2 (2001), pp. 229–233.

[62] Ammar Jalalimanesh et al. "Simulation-based optimization of radiotherapy: Agent-based modeling and reinforcement learning". In: *Mathematics and Computers in Simulation* 133 (2017), pp. 235–248.

[63] Mohammad Ali Javidian, Pooyan Jamshidi, and Marco Valtorta. "Transfer Learning for Performance Modeling of Configurable Systems: A Causal Analysis". In: *arXiv preprint arXiv:1902.10119* (2019).

[64] Kristine Jones et al. "Cost-effectiveness of critical time intervention to reduce homelessness among persons with mental illness". In: *Psychiatric Services* 54.6 (2003), pp. 884–890.

[65] Michael J. Kane et al. "Comparison of ARIMA and Random Forest time series models for prediction of avian influenza H5N1 outbreaks". In: *BMC Bioinformatics* 15.1 (2014). ISSN: 14712105. DOI: 10.1186/1471-2105-15-276.

[66] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2018. arXiv: 1812.04948 [cs.NE].

[67] Tero Karras et al. "Analyzing and improving the image quality of stylegan". In: *arXiv preprint arXiv:1912.04958* (2019).

[68] Thomas N. Kaye and David A. Pyke. "The effect of stochastic technique on estimates of population viability from transition matrix models". In: *Ecology* 84.6 (2003), pp. 1464–1476. ISSN: 00129658. DOI: 10.1890/0012-9658(2003)084[1464:TEOSTO]2.0.CO;2.

[69] Nico Keilman. "Erroneous Population Forecasts". In: *Old and New Perspectives on Mortality Forecasting*. Ed. by Tommy Bengtsson and Nico Keilman. Cham: Springer International Publishing, 2019. Chap. 9, pp. 95–111. ISBN: 978-3-030-05075-7. DOI: 10.1007/978-3-030-05075-7_9.

[70] Amin Khademi et al. "An Agent-Based Model of Healthy Eating with Applications to Hypertension". In: *Advanced Data Analytics in Health*. Springer, 2018, pp. 43–58.

[71] Pedram Khayyatkhoshnevis et al. "Smart City Response to Homelessness". In: *IEEE Access* 8 (2020), pp. 11380–11392.

[72] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[73] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[74] Adam J Kunen, Teresa S Bailey, and Peter N Brown. *KRIPKE-a massively parallel transport mini-app*. Tech. rep. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2015.

[75] Bogdan Kustowski et al. *Transfer Learning as a Tool for Reducing Simulation Bias: Application to Inertial Confinement Fusion*. Tech. rep. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2019.

[76] Farha L. *We can't talk about inequality without talking about homelessness; in The Guardian. International Edition*. 2016.

[77] Darius Lam et al. "xview: Objects in context in overhead imagery". In: *arXiv preprint arXiv:1802.07856* (2018).

[78] E Latimer et al. *Dénombrement des personnes en situation d'itinérance à Montréal le 24 mars 2015/I Count Montreal: Count and Survey of Montreal's Homeless Population on March 24, 2015*. 2015.

[79] Eric A Latimer et al. "Costs of services for homeless people with mental illness in 5 Canadian cities: a large prospective follow-up study". In: *CMAJ open* 5.3 (2017), E576.

[80] Eric A Latimer et al. "Costs of services for homeless people with mental illness in 5 Canadian cities: a large prospective follow-up study". In: *CMAJ open* 5.3 (2017), E576.

[81] Latimer E, Rabouin D, Méthot C, McAll C, Ly A, Dorvil H, et al. "At Home/Chez Soi Project: Montréal Site Final Report". In: *Calgary, AB: Mental Health Commission of Canada* (2014).

[82] Bruno Lepri et al. "The role of personality in shaping social networks and mediating behavioral change". In: *User Modeling and User-Adapted Interaction* 26.2-3 (2016), pp. 143–175. ISSN: 15731391. DOI: 10.1007/s11257-016-9173-y.

[83] Andy Liaw and Matthew et al. Wiener. "Classification and regression by random forest". In: *R news* 2.3 (2002), pp. 18–22.

[84] Weiyang Liu et al. "Large-margin softmax loss for convolutional neural networks." In: *ICML*. Vol. 2. 3. 2016, p. 7.

[85] Davey M. and Knaus C. *Homelessness in Australia up 14% in five years, ABS says; in The Guardian*. 2018.

[86] Mago, Vijay K and Morden, Hilary K and Fritz, Charles and Wu, Tiankuang and Namazi, Sara and Geranmayeh, Parastoo and Chattopadhyay, Rakhi and Dabbaghian, Vahid. "Analyzing the impact of social factors on homelessness: a Fuzzy Cognitive Map approach". In: *BMC medical informatics and decision making* 13.1 (2013), p. 94.

[87] Preeti Malakar et al. "Benchmarking Machine Learning Methods for Performance Modeling of Scientific Applications". In: *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE. 2018, pp. 33–44.

[88] Mansur, Erin and Quigley, John and Raphael, Steven and Smolensky, Eugene. "Examining policies to reduce homelessness using a general equilibrium model of the housing market". In: *Journal of Urban Economics* 52.2 (2002), pp. 316–340.

[89] Aniruddha Marathe et al. "Performance modeling under resource constraints using deep transfer learning". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM. 2017, p. 31.

[90] Kirsten Martin. "Ethical Implications and Accountability of Algorithms". In: *Journal of Business Ethics* 160.4 (2019), pp. 835–850. ISSN: 15730697. DOI: 10.1007/s10551-018-3921-3. URL: http://dx.doi.org/10.1007/s10551-018-3921-3.

[91] Michelle McKenzie et al. "Tracking and follow-up of marginalized populations: A review". In: *Journal of Health Care for the Poor and Underserved* 10.4 (1999), pp. 409–429. ISSN: 10492089. DOI: 10.1353/hpu.2010.0697.

[92] Meghan Henry et al. *The 2014 Annual Homelessness Assessment Report to Congress, Part 1: Point-in-time estimates of homelessness*. Tech. rep. 2014.

[93] P. A. Moran and Peter Whittle. "Hypothesis Testing in Time Series Analysis." In: *Journal of the Royal Statistical Society. Series A (General)* 114.4 (1951), p. 579. ISSN: 00359238. DOI: 10.2307/2981095.

[94] Geoffrey Nelson et al. "Life Changes Among Homeless Persons With Mental Illness: A Longitudinal Study of Housing First and Usual Treatment". In: *Psychiatric Services in Advance* 27.1 (2015), pp. 1–6. ISSN: 10752730. DOI: 10.1176/appi.

[95] Kirsi Nousiainen. "Reflecting narrative interview context as performance: interviews with former homeless persons with intoxication and mental health problems". In: *Nordic Social Work Research* 5.2 (2015), pp. 129–142. ISSN: 2156-857X. DOI: 10.1080/2156857X.2015.1042018. URL: http://www.tandfonline.com/doi/full/10.1080/2156857X.2015.1042018.

[96] O'Flaherty, B. "Individual homelessness: Entries, exits, and policy". In: *Journal of Housing Economics* 21 (2012), pp. 77–100.

[97] Donna H. Odierna and Laura A. Schmidt. "The effects of failing to include hard-to-reach respondents in longitudinal surveys". In: *American Journal of Public Health* 99.8 (2009), pp. 1515–1521. ISSN: 00900036. DOI: 10.2105/AJPH.2007.111138.

[98] Deborah K Padgett, Leyla Gulcur, and Sam J Tsemberis. "Housing First Services for People Who Are Homeless With Co-Occurring Serious Mental Illness and Substance Abuse". In: *Research on Social Work Practice* 16 (2006), pp. 74–83. ISSN: 1049-7315. DOI: 10.1177/1049731505282593.

[99] Barak A. Pearlmutter. "Learning state space trajectories in recurrent neural networks". In: *IJCNN Int Jt Conf Neural Network* (1989), pp. 365–372. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.2.263.

[100] Myra Piat et al. "Pathways into homelessness: Understanding how both individual and structural factors contribute to and sustain homelessness in Canada". In: *Urban Studies* 52.13 (2015), pp. 2366–2382.

[101] Alessandro Piscopo, Ronald Siebes, and Lynda Hardman. "Predicting Sense of Community and Participation by Applying Machine Learning to Open Government Data". In: *Policy and Internet* 9.1 (2017), pp. 55–75. ISSN: 19442866. DOI: 10.1002/poi3.145.

[102] Martijn Poel, Eric T. Meyer, and Ralph Schroeder. "Big Data for Policymaking: Great Expectations, but with Limited Progress?" In: *Policy and Internet* 10.3 (2018), pp. 347–367. ISSN: 19442866. DOI: 10.1002/poi3.176.

[103] Michael JD Powell. "The theory of radial basis function approximation in 1990". In: *Advances in numerical analysis* (1992), pp. 105–210.

[104] Hannah Rae Rabinovitch. "Snapshot of an object in motion: quantifying homelessness". Master. Simon Fraser University, 2015. URL: http://summit.sfu.ca/item/15297.

[105] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[106] Gernot Rieder and Judith Simon. "Big Data: A New Empiricism and its Epistemic and Socio-Political Consequences". In: *Berechenbarkeit der Welt?* (2017), pp. 85–105. DOI: 10.1007/978-3-658-12153-2_4.

[107] Martin Riedmiller. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *European Conference on Machine Learning*. Springer. 2005, pp. 317–328.

[108] Stewart Robinson. *Simulation: The Practice of Model Development and Use*. Palgrave Macmillan, 2014.

[109] Jill S Roncarati et al. "Mortality among unsheltered homeless adults in Boston, Massachusetts, 2000-2009". In: *JAMA internal medicine* 178.9 (2018), pp. 1242–1248.

[110] Laurence Roy et al. "Criminal behavior and victimization among homeless individuals with severe mental illness: a systematic review". In: *Psychiatric services* 65.6 (2014), pp. 739–750.

[111] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[112] Jerome Sacks, Susannah B Schiller, and William J Welch. "Designs for computer experiments". In: *Technometrics* 31.1 (1989), pp. 41–47.

[113] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2018. arXiv: 1801.04381 [cs.CV].

[114] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[115] Ramesh Sharda and Rajendra B. Patil. "Connectionist approach to time series prediction: an empirical test". In: *Journal of Intelligent Manufacturing* 3.5 (1992), pp. 317–323. ISSN: 09565515. DOI: 10.1007/BF01577272.

[116] Donald Shepard. "A two-dimensional interpolation function for irregularly-spaced data". In: *Proceedings of the 1968 23rd ACM national conference*. ACM. 1968, pp. 517–524.

[117] Kristin Spröhnle, Eva-Maria Fuchs, and Patrick Aravena Pelizari. "Object-based analysis and fusion of optical and SAR satellite data for dwelling detection in refugee camps". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 10.5 (2017), pp. 1780–1791.

[118] Judith A. Stein and Adeline M. Nyamathi. "Completion and Subject Loss Within an Intensive Hepatitis Vaccination Intervention Among Homeless Adults: The Role of Risk Factors, Demographics, and Psychosocial Variables". In: *Health Psychology* 29.3 (2010), pp. 317–323. ISSN: 02786133. DOI: 10.1037/a0019283.

[119] Stergiopoulos V, Hwang SW, Gozdzik A, Nisenbaum R, Latimer E, Rabouin D, et al. "Effect of scattered-site housing using rent supplements and intensive case management on housing stability among homeless adults with mental illness: a randomized trial". In: *JAMA* 313(9) (2015), pp. 905–15.

[120] John Sterman. "Business Dynamics: Systems Thinking and Modeling for a Complex World". In: (2000).

[121] Maurice E Stucke. "Big data and competition law". In: *Concurrences* 2017.4 (2017). ISSN: 21160090. DOI: 10.15375/zwer-2017-0405.

[122] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[123] Lancet The. "A shared future for all: let's talk about homelessness." In: *Lancet (London, England)* 391.10117 (2018), p. 179.

[124] Matt Thomson. *Vancouver Homeless Count 2015*. M. Thomson Consulting, 2015.

[125] City of Toronto. *Street Needs Assessment Results 2013*. 2013.

[126] Sam J Tsemberis et al. "Measuring Homelessness and Residential Stability: the Residential Time-Line Follow-Back Inventory". In: *Journal of Community Psychology* 35.1 (2007), pp. 29–42.

[127] Tsemberis, S. "Housing First: The Pathways Model to End Homelessness for People with Mental Illness and Addiction Manual". In: *Center City: Hazelden* (2010).

[128] Alexis Tsoukias et al. "Policy analytics: An agenda for research and practice". In: *EURO Journal on Decision Processes* 1.1-2 (2013), pp. 115–134. ISSN: 21939446. DOI: 10.1007/s40070-013-0008-3.

[129] City of Vancouver. *Vancouver Homeless Count 2018*. 2018.

[130] Mago VK et al. "Analyzing the impact of social factors on homelessness: A Fuzzy Cognitive Map approach". In: *BMC: Medical Informatics and Decision-Making* 13(94) (2013).

[131] Shifeng Wang, Emily So, and Pete Smith. "Detecting tents to estimate the displaced populations for post-disaster relief using high resolution satellite imagery". In: *International Journal of Applied Earth Observation and Geoinformation* 36 (2015), pp. 87–93.

[132] Jennifer Ward and Zoë Henderson. "Some practical and ethical issues encountered while conducting tracking research with young people leaving the 'care' system". In: *International Journal of Social Research Methodology: Theory and Practice* 6.3 (2003), pp. 255–259. ISSN: 13645579. DOI: 10.1080/1364557032000091851.

[133] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[134] Izaak L. Williams and Clifford R. O'Donnell. "Web-based tracking methods in longitudinal studies". In: *Evaluation and Program Planning* 45 (2014), pp. 82–89. ISSN: 01497189. DOI: 10.1016/j.evalprogplan.2014.04.001. URL: http://dx.doi.org/10.1016/j.evalprogplan.2014.04.001.

[135] Jennifer R Wolch and Stacy Rowe. "On the Streets: Mobility Paths of the Urban Homeless". In: *City & Society* 6.2 (1992), pp. 115–140.

[136] George L Wolford et al. "Evaluation of methods for detecting substance use disorder in persons with severe mental illness." In: *Psychology of Addictive Behaviors* 13.4 (1999), pp. 313–326. ISSN: 1939-1501(Electronic),0893-164X(Print). DOI: 10.1037/0893-164X.13.4.313.

[137] Kathleen W Wyrwich et al. "Measuring patient and clinician perspectives to evaluate change in health-related quality of life among patients with chronic obstructive pulmonary disease." In: *Journal of general internal medicine* 22.2 (Feb. 2007), pp. 161–70. ISSN: 1525-1497. DOI: 10.1007/s11606-006-0063-6.

[138] Bing Xu et al. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).

[139] Zhang, T., Xie, S., & Rose, O. "Real-time batching in job shops based on simulation and reinforcement learning". In: *2018 Winter Simulation Conference* (2018), pp. 3331–3339.

Appendix A

Boxes

```
def cf_error(surrogate, split_set, x_data, y_data, predictors):
        predictions = [0.0] * len(predictors)
        rmse = 0.0
        count = 0
        for train_index, test_index in split_set:
                # Split the data
                X_train, X_test = x_data[train_index], x_data[test_index]
                Y_train, Y_test = y_data[train_index], y_data[test_index]
                # Train the model
                temp = copy_model(surrogate)
                temp.set_training_values(X_train, Y_train)
                temp.train()
                # Get the RMSE
                cur_rmse = float(compute_rms_error(temp, X_test, Y_test))
                rmse += cur_rmse
                count += 1
                # Get predictions
                yp = list(temp.predict_values(predictors))
                for i in range(0, len(yp)):
                    predictions[i] += yp[i]
        # Average the predictions
        for i in range(0, len(predictions)):
            predictions[i] /= float(count)
        return (rmse / float(count)), predictions
```

Box 1. Python code to validate a machine learning model on a dataset using 10-cross fold validation.

Appendix B

Algorithms

Algorithm 1 Homelessness Forecasting.

Input: A = the overall population, t = the current week in the simulation
Output: The *loss* of this epoch- that is, how much the transition probabilities were changed.

procedure SIMULATION(A, t)
    $tr\_mat \leftarrow Q_t$
    $tr\_counts \leftarrow copy\_structure(tr\_mat)$
    $mdq\_probs \leftarrow copy\_structure(tr\_mat)$
    $mnfq\_rewards \leftarrow copy\_structure(tr\_mat)$
    for each $p \in A_t$ **do**
        $s_i = s_{p,t}$ State of individual p at time t
        $s_n \leftarrow Wheel(tr\_mat[s_i])$
        $s_{p,t+1} = s_n$ State of individual p at next time step
        $q_{p,t} \leftarrow mdq\_nn.process(s_i, s_n)$
        $mnfq\_nn \leftarrow mnfq\_nns[s_i][s_n]$
        $\hat{n}_{j,p,t}, lowest\_error \leftarrow$
          $mnfq\_nn.process(curr\_mdq\_prob, \hat{N}_t, s_n, 1)$
        $mnfq\_rewards[s_i][s_n] \mathrel{+}= curr\_mnfq\_reward$
        if $lowest\_error == True$ **then**
            $mdq\_prob[s_i][s_n] = curr\_mdq\_prob$
        end if
        $tr\_counts[s_i][s_n] \mathrel{+}= 1$
    end for
    $mnfq\_rewards = mean(mnfq\_rewards, tr\_counts)$
    $new\_tr\_mat =$
          $update\_prob(mdq\_prob, mnfq\_rewards, tr\_mat)$
    $loss = mean\_difference(new\_tr\_mat, tr\_mat)$
    $accuracy = difference(A_{t+1})$
    $transition\_matrices[t] = new\_tr\_mat$
    return $loss, accuracy$
end procedure

Algorithm 2 Roulette Wheel.

Input: *possible_transitions* = the possible transitional probabilities to consider
Output: s_n = the new state to transition to

```
1: procedure WHEEL(possible_transitions)
2:     s_n ← 0
3:     random_state = random[0, 1]
4:     sum ← next(possible_transitions)
5:     while random_state ≥ sum do
6:         sum += next(possible_transitions)
7:         s_n += 1
8:     end while
9:     return s_n
10: end procedure
```

Algorithm 3 DCGAN Generator Initialization Method.

```
1: model = SequentialModel()
2: layers = array()
3:
4: layers.add(Dense(use_bias_parameter))
5: layers.add(BatchNormalization())
6: layers.add(Activation_Function())
7:
8: layers.add(Convolution_Transpose(128, use_bias_parameter))
9: layers.add(Batch_Normalization())
10: layers.add(Activation_Function())
11:
12: layers.add(Convolution_Transpose(64, use_bias_parameter))
13: layers.add(Batch_Normalization())
14: layers.add(Activation_Function())
15:
16: layers.add(Convolution_Transpose(use_bias_parameter))
17:
18: optimizer = Optimizer()
19: model.create(layers, optimizer)
20: return model, optimizer
```

Algorithm 4 DCGAN Discriminator Initialization Method.

```
1: model = SequentialModel()
2: layers = array()
3:
4: layers.add(Convolution(64))
5: layers.add(Activation_Function())
6: layers.add(Dropout_Layer(dropout_percentage))
7:
8: layers.add(Convolution(128))
9: layers.add(Activation_Function())
10: layers.add(Dropout_Layer(dropout_percentage))
11:
12: layers.add(Dense(64))
13:
14: optimizer = Optimizer()
15: model.create(layers, optimizer)
16: return model, optimizer
```

Appendix C

Abbreviations

Abbreviation	**Description**
MDQL	Modified Deep Q-Learning
MNFQ	Modified Neural Fitted Q-Iteration
SIR	Susceptible Infectious Recovered
ICU	Intensive Care Unit
SUS	System Usability Scale
ABM	Agent-Based Model
IDW	Inverse Distance Weighting
RBF	Radial Basis Function
QP	Quadratic Polynomials
LS	Least Squares
SMT	Surrogate Modeling Toolbox
RMTB	Regularized Minimal-energy Tensor-product B-splines
RMTC	Regularized Minimal-energy Tensor-product Cubic hermite splines
RMSE	Root-Mean-Squared-Error
ANN	Artificial Neural Networks
HF	Housing First
TAU	Treatment As Usual
PIT	Point-In-Time
ARIMA	AutoRegressive Integrated Moving Average
GRU	Gated Recurrent Unit

LSTM	Long-Short Term Memory
RNN	Recurrent Neural Network
MLP	MultiLayer Perceptron
WMAE	Weighted Mean Absolute Error
CA	Cellular Automata
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network

Appendix D

Resources

Description	**Link**
A demonstration of the first iteration of the homelessness simulation platform for a presentation at the fourth annual Canadian Homeless Data Sharing initiative "Data That Makes a Difference" in 2019	`https://www.youtube.com/watch?v=dqHiug0OUpA`
A tutorial on how to develop an API using the Python library "Flask" which was used in development of the homelessness simulation platform	`https://www.youtube.com/watch?v=opKls658yUU`
A tutorial on how to use the HPC at Lakehead University which was an integral part in providing computational power during my studies	`https://www.youtube.com/watch?v=KLwfrzYjdM0`
A presentation of TentNet at IEEE SMC in 2020	`https://vimeo.com/showcase/7648425/video/466038898`
A repository of the codebase used in Chapter 3	`https://github.com/andrfish/MDQL-with-MNFQ`