Deep Learning Models for Route Planning in Road Networks

Thesis · June 2018

DOI: 10.13140/RG.2.2.30527.56486

Tianyu Zhou
KTH Royal Institute of Technology
4 PUBLICATIONS **2** CITATIONS

DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Deep Learning Models for Route Planning in Road Networks

TIANYU ZHOU

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Abstract

Traditional shortest path algorithms can efficiently find the optimal paths in graphs using simple heuristics. However, formulating a simple heuristic is challenging under the road network setting since there are multiple factors to consider, such as road segment length, edge centrality, and speed limit. This study investigates how a neural network can learn to take these factors as inputs and yield a path given a pair of origin and destination. The research question is formulated as: *Are neural networks applicable to real-time route planning tasks in a road network?*. The proposed metric to evaluate the effectiveness of the neural network is *arrival rate*. The quality of generated paths is evaluated by *time efficiency*. The real-time performance of the model is also compared between pathfinding in dynamic and static graphs, using the above metrics.

A staggered approach is applied in progressing this investigation. The first step is to generate random graphs, which allows us to monitor the size and properties of the training graph without caring too many details in a road network. The next step is to determine, as a proof of concept, if a neural network can learn to traverse simple graphs with multiple strategies, given that road networks are in effect complex graphs. Finally, we scale up by including factors that might affect the pathfinding in real road networks. Overall, the training data is optimal paths in a graph generated by a shortest path algorithm. The model is then applied to new graphs to generate a path given a pair of origin and destination. The *arrival rate* and *time efficiency* are calculated and compared with that of the corresponding optimal path.

Experimental results show that the effectiveness, i.e., *arrival rate* of the model is 90% and the path quality, i.e., *time efficiency* has a median of 0.88 and a large variance. The experiment shows that the model has better performance in dynamic graphs than in static graphs. Overall, the answer to the research question is positive. However, there is still room to improve the effectiveness of the model and the paths generated by the model. This work shows that a neural network trained to make locally optimal choices can hardly give a globally optimal solution. We also show that our method, only making locally optimal choices, can adapt to dynamic graphs with little performance overhead.

Keywords: Route Planning, Pathfinding, Shortest Path Algorithms, Road Networks, Deep Learning, Neural Networks.

Sammanfattning

Traditionella algoritmer för att hitta den kortaste vägen kan effektivt hitta de optimala vägarna i grafer med enkel heuristik. Att formulera en enkel heuristik är dock utmanande för vägnätverk eftersom det finns flera faktorer att överväga, såsom vägsegmentlängd, kantcentralitet och hastighetsbegränsningar. Denna studie undersöker hur ett neuralt nätverk kan lära sig att ta dessa faktorer som indata och finna en väg utifrån start- och slutpunkt. Forskningsfrågan är formulerad som: *Är neuronnätverket tillämpliga på realtidsplaneringsuppgifter i ett vägnät?*. Det föreslagna måttet för att utvärdera effektiviteten hos det neuronnätverket är *ankomstgrad*. Kvaliteten på genererade vägar utvärderas av *tidseffektivitet*. Prestandan hos modellen jämförs också mellan sökningen i dynamiska och statiska grafer, med hjälp av ovanstående mätvärden.

Undersökningen bedrivs i flera steg. Det första steget är att generera slumpmässiga grafer, vilket gör det möjligt för oss att övervaka träningsdiagrammets storlek och egenskaper utan att ta hand om för många detaljer i ett vägnät. Nästa steg är att, som ett bevis på konceptet, undersöka om ett neuronnätverk kan lära sig att korsa enkla grafer med flera strategier, eftersom vägnätverk är i praktiken komplexa grafer. Slutligen skalas studien upp genom att inkludera faktorer som kan påverka sökningen i riktiga vägnät. Träningsdata utgörs av optimala vägar i en graf som genereras av en algoritm för att finna den kortaste vägen. Modellen appliceras sedan i nya grafer för att hitta en väg mellan start- och slutpunkt . *Ankomstgrad* och *tidseffektivitet* beräknas och jämförs med den motsvarande optimala sökvägen.

De experimentella resultaten visar att effektiviteten, dvs ankomstgraden av modellen är 90% och vägkvaliteten dvs tidseffektiviteten har en median på 0,88 och en stor varians. Experimentet visar att modellen har bättre prestanda i dynamiska grafer än i statiska grafer. Sammantaget är svaret på forskningsfrågan positivt. Det finns dock fortfarande utrymme att förbättra modellens effektivitet och de vägar som genereras av modellen. Detta arbete visar att ett neuronnätverk tränat för att göra lokalt optimala val knappast kan ge globalt optimal lösning. Vi visar också att vår metod, som bara gör lokalt optimala val, kan anpassa sig till dynamiska grafer med begränsad prestandaförlust.

Nyckelord: Ruttplanering, Sökning, Algoritmer för att finna kortaste väg, Vägnät, Djupinlärning, Neuronnätverk.

Acknowledgements

I am grateful to those who helped me during the entire journey of thesis work. I want to say thank you to Douglas Garcia Torres, Tongtong Fang, and Hongyi Liu who read my draft thesis report and gave critical and constructive comments and suggestions. I need to thank my host company, LakeTide AB with all the support, especially the patient guidance I obtained from my industrial supervisor, Robert Luciani. I want to express my special gratitude to my examiner, Professor Henrik Boström, and supervisor, Assoc. Professor Sarunas Girdzijauskas, who have been patiently guiding my thesis writing process.

Tianyu Zhou

June 20, 2018

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Route planning, i.e., pathfinding, in road networks aims to find the optimal paths between a pair of specified origin and destination [2]. It serves as a guideline for subjects such as traffic engineering and transportation planning. Route planning is also widely used by various transportation agencies, academic institutions, and enterprises. These organizations plan routes to either assist their management of the road networks or provide corresponding services. Route planning in road networks faces a few challenges.

Firstly, the standards of an optimal path can be various. For road network traveling, some people prefer minimum time; some prefer minimum traveling distance; some want to make as few changes of roads as possible; some try to avoid urban arterial roads or highway, or the other way around. It is also possible that the combination of those criteria needs consideration. Secondly, the expansion of modern cities and road networks makes the route planning more and more computationally expensive. Thirdly, a road network, as a dynamic system, makes it difficult for traditional route planning, which treats the road network as static. This makes real-time route planning challenging in road networks. Finally, the state of a road network is determined by many factors such as road segment length, speed limit, current road condition, and road popularity. Traditional algorithms require much engineering effort to take into account if not all, but most of those factors.

Neural network recently has grown as one of the most popular machine learning models. Provided with a considerable amount of training data, Neural networks are able to learn to perform well on various tasks [3]. It is reasonable to speculate that a neural network can be potentially trained to traverse a road network, given properties of a road network such road length, speed, and centrality as input. The effectiveness of the trained model is measured by *arrival rate*, i.e., given a pair of origin and destination, how likely does the model give a path. Ideally, the *arrival rate* should be 100%. The path quality should also be guaranteed, i.e., the *time efficiency* should be as close to the optimal path as possible. It is indisputable that there are many other machine learning algorithms that could be suitable such as Support Vector Machine (SVM) [4], Decision Tree (DT) [5], and Random Forest (RF) [6]. SVM, RF, and DT have been successful in the industry for many years [7].

This thesis work will be centered around neural network models.

1.2 Problem

The most commonly used algorithm for pathfinding is Dijkstra's algorithm [8], which has the time complexity of $O(E+V \log V)$ where E and V are the number of nodes (crossroads) and number of edges (road segments in a road network). Dijkstra's algorithm can find the optimal path but it is not scalable to large road networks with thousands or millions of conjunctions [9, 10, 11, 12]. Its most popular variation, A* search algorithm [13], uses heuristics to speed up the search and has the time complexity of $O(E)$ [14]. In the worst case, A* search has the same time complexity as Dijkstra's algorithm [13]. Traditional shortest path algorithms fall short of capturing the dynamic essence of road networks. As pointed out by Garg et al. [15], dynamic graph problems are more challenging and time consuming as compared to the static ones. Dynamic shortest path algorithms have been proposed to overcome this challenge [16, 17, 18, 19, 20, 15]. However, as concluded by E. N. Tamatjita et al. [20], it is not feasible to use Dijkstra's algorithm on a dynamic road network that consists of bidirectional roads. This severely limits the use of Dijkstra's algorithm in road networks.

A* search algorithm requires the heuristic to be admissible, meaning it never overestimates [14]. Commonly used heuristics are Manhattan Distance [21], Diagonal Distance [22], and Euclidean Distance [23]. There is no heuristic designed to combine many factors such as road segment properties and traffic conditions. Furthermore, under the road network setting, is the optimal path always necessary? Perhaps an algorithm that gives good-enough paths yet more computationally efficient is more favored. A neural network-enabled algorithm makes it potential for the pathfinding to be more dynamic and flexible, taking into account road network properties, road conditions, and other factors that typically affect road network states.

Neural network approaches for route planning in road networks have been applied in many studies. The study from Alex et al. [24] shows that neural networks can indeed be trained to perform pathfinding tasks. The trained model executed traversal and shortest path tasks, yielding an average accuracy of 98.8% on traversal tasks and an average accuracy of 55.3% on shortest path tasks. However, the complexity of traversal tasks does not reflect the complexity of real road networks. The traversal tasks are constrained by four-step traversal and shortest path tasks are limited under seven-step paths in the map. An important work that sets ground for this thesis work is from Xu et al. [25]. They used road segment length, path saturation [26], and vehicle speed to generate weights for each road. Then a Pulse Coupled Neural Network (PCNN) [27] was further applied to select the optimal path according to the weights. However, this work only considered three variables. There are more factors such as centrality, distance, and direction could be included. Moreover, the experiment of this study is on an undirected graph with 16 nodes. The algorithm proposed in this study involves a large number of matrix operations, making it potentially non-scalable to large graphs. Zhang et al. [28] also used PCNN for all pairs shortest path problem and they experimented on a graph with 8 nodes and 24 edges. This study relies heavily on matrix multiplication, which is computationally intensive and might also hinder its scalability to large graphs. The above two studies both presented oversimplified experiments. For applications in a road network, it usually contains thousands of nodes or more. These studies also lack experiments on the real-time route planning tasks, which is also a common setting in road networks. Furthermore, these three studies used relatively complex neural network architectures, it is of interest to investigate if simple neural network architecture can be applicable so that it is easier to study and reproduce.

1.3 Purpose

Traditional shortest path algorithms such as A* search have been proven to be effective and efficient. Still, they have difficulties handling the dynamics of a road network. Past neural network approaches for route planning did not consider the dynamics of a graph and they might not be scalable to large graphs. The knowledge gap remains to investigate if a neural network model can be trained to handle the dynamics of a road network containing thousands of nodes. Therefore, the research question is formulated as the following:

Are neural networks applicable to real-time route planning tasks in a road network?

The effectiveness of the neural network is evaluated with *arrival rate* (section 3.3.1). The quality of generated paths is evaluated by *time efficiency* (section 3.3.2) and compared with that of the optimal paths. The real-time pathfinding of the model is evaluated by comparing the measurements in dynamic and static graphs, using the above metrics.

1.4 Objectives

In this study, we want to use neural networks to perform pathfinding tasks so that it is real-time and scalable. To bridge the knowledge gap, the following objectives are set:

1. We generate random graphs with customized algorithms and training data with existing shortest path algorithms; neural networks are trained and evaluated on new random graphs.
2. We design an algorithm to randomly modify the edge weights to evaluate how the model performs, using the proposed metrics.
3. Models trained on random graphs are qualitatively evaluated on a road network from OpenStreetMap (OSM).

1.5 Methodology

A wide literature study is conducted to gain knowledge about the domain of shortest path algorithms and neural networks. To answer the research question, the *empirical research* [29] method is adopted. We first design a procedure to generate random graphs. This allows us to monitor the size and properties of the training graph without caring too many details in a road network. Neural network models are trained on the data generated from a random graph and evaluated quantitatively on new graphs. This is how we test the scalability of the model. To see if the model functions in a dynamic graph, edge weights are modified randomly. Finally, we apply the trained model on a real road network downloaded from OSM. The paths generated by the model is qualitatively evaluated with the corresponding optimal paths. Therefore, in this study, both quantitative and qualitative [29, 30] methods are applied to evaluate the neural network models. More details can be seen in section 3.

1.6 Ethics, Benefits & Sustainability

This thesis work does not use any private data. All data is generated with procedures specifically implemented for this thesis. Therefore, there is no ethic issues involved. With the fast expansion of cities and urban areas, route planning has become more and more important for urban economic development and sustainability. There is an urgent need for better inner- and inter-city route planning. Having a computational model that has low time complexity and is able to guarantee path quality can be beneficial to various organizations such as governments, transportation agencies, and academic institutions. The societal benefit of this study aligns with the ninth and eleventh Sustainable Development Goal [1], which are *INDUSTRY, INNOVATION AND INFRUSTRUCTURE* and *SUSTAINABLE CITIES AND COMMUNITIES*. The positive effects include but are not limited to better public transportation, better inner-city and inter-city traffic condition, better energy efficiency.

1.7 Delimitations

The road network used in this work is taken from OSM with only the Stockholm area. Other road networks from other sources are not considered. This work does not aim to compare which machine learning algorithms are more suitable for pathfinding in road networks. The neural network model is the only one investigated in this study. The performance of the model in different hardware settings is out of the scope.

1.8 Outline

The rest of the thesis is organized as the following:

- *Extended Background* will have further discussion of shortest path algorithms, and neural network theories.
- *Methodology* elaborates the design choices, the approaches used, and steps taken to answer the research question.
- *Algorithm Implementation* describes a set of algorithms implemented in order to answer the research question.

[1] https://www.un.org/sustainabledevelopment/sustainable-development-goals/

- *Results* presents the experimental settings and procedures. Corresponding results are presented and further elaborated.
- *Conclusion* will conclude this study and provide a further discussion about the results. Findings and insights will be shared.

2 Extended Background

2.1 Graph Traversal

Route planning can be categorized into a bigger top topic, graph traversal. Graph traversal has been well studied in the last 50 years. In computer science, graph traversal is the process of visiting each node in a graph. Based on the order of visiting nodes, graph traversal can be divided into two main categories, depth-first search [31] and breadth-first search [32]. Both have been applied to solve many important problems. Best-first search is also an important traversal strategy. Best-first algorithms traverse a graph by following the most “promising” nodes according to specific rules. Those rules are often called heuristics [33].

Heuristics are commonly used to speed up the search algorithms. Heuristics are also applied when the original algorithms fail to find the optimal solutions or when it is not necessary to find the best ones. A heuristic finds an approximate solution that is good enough yet much faster. Heuristics guide the search by trying to be ”smart” about how close a node is away from the destination [14, 34]. “Closer” nodes are explored first.

It is rare that people drive aimlessly, meaning they usually have a destination. Meantime, people want to arrive their destinations fast. This makes shortest path algorithms useful not only in real life but also the center of this work. Online map service providers such as Google Map [2] and Bing Map [3] rely on shortest path algorithms [35]. In this study, we use the output of shortest path algorithms as the ground truth when training a neural network and the benchmark for evaluation. Heuristics guided shortest path algorithms will be extensively discussed and applied to guide the training of the neural network.

2.2 Shortest Path Algorithms

Shortest path algorithms aim to find a path between a pair of nodes such that the sum of edge weights are minimized [36]. Floyd-Warshall, Bellman-Ford, Dijkstra’s, and A* search are the most popular algorithms for finding the

[2]https://www.google.com/maps
[3]https://www.bing.com/maps

shortest paths. Floyd-Warshall algorithm [37] finds the shortest paths for all pairs of nodes in a graph. This is often overkill and suffers from the high computational cost ($\Theta(V^3)$ where V is the number of nodes). Bellman-Ford algorithm [38] is a single source shortest path algorithm that can handle negative edge weights with time complexity of $\Theta(VE)$ where E is the number of edges. This is a big advantage in many scenarios. However, in this study, we limit the scope in the case where the edge weights of road networks are positive real numbers. Moreover, the implementation of Bellman-Ford shares much resemblance with Dijkstra's. It is unnecessary to extensively explore this algorithm and it is reasonable to focus on a smaller set of algorithms.

Dijkstra's algorithm [8] finds shortest paths from one source to all other nodes. It is unable to handle negative edge weights and this is acceptable in this study. Dijkstra's algorithm has the time complexity of $O(E + V \log V)$. A* search algorithm [13] is one of the best-first search algorithms. It has even been more widely used than Dijkstra in many scenarios due to its performance and accuracy. As an extension of Dijkstra's algorithm, A* search uses customized heuristics to guide the search. In theory, A* search has better time complexity ($O(E)$) than Dijkstra's.

The goal of Dijkstra' s and A* search algorithms is similar mathematically,

$$f(n) = g(n) + h(n)$$

where n is the current node that is being explored, $g(n)$ is the total weights of the path from the starting point to node n, and $h(n)$ is a customized heuristic 2.1 function that estimates the cost from n to the destination. Therefore, Dijkstra's and A* search try to find a path that minimize $f(n)$. For Dijkstra's algorithm, $h(n)$ is always zero, meaning it does not try to be smart hence less efficiency. It can be expected that the implementation of the two algorithms is based on the same structure, as shown in appendix A. The use of heuristics are problem-specific and might affect the convergence of the algorithm. To guarantee the convergence of A* search algorithm, meaning always find a path, the heuristic calculation should be admissible [39]. A heuristic being admissible simply means it never overestimates the distance to the destination. If $h^*(n)$ denotes the optimal cost to the destination from n, then $h(n)$ is admissible if: $\forall n, h(n) \leq h^*(n)$.

2.3 Neural Networks

Neural network is a computational model dates back to the late 1950s, which was roughly inspired by biological neural systems. Deep neural networks have been applied in various fields not only those that were difficult to solve with traditional machine learning algorithms, image classification for instance but also relatively new fields such as bioinformatics and language modeling. The earliest implementation of neural network had a different name, perceptron.

2.3.1 Perceptron

The perceptron was invented by Frank Rosenblatt in 1957 [40] as a simplified imitation of biological neuron. Mathematically speaking, the perceptron algorithm was an approximator of a binary classifier. Stated differently, it is a function that maps an input vector x to an binary output y such as the following:

$$y = f(x) = \begin{cases} 1 & w \cdot x + b > 0 \\ 0 & otherwise \end{cases}$$

where w is vector of real-valued weights and b is the bias, seen from figure 1. This computation is essentially an affine transformation [4].

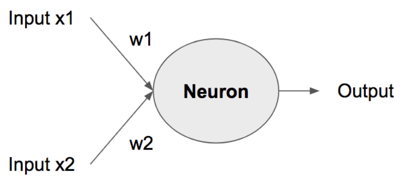


Figure 1: An illustration of Perceptron.

Stated in geometric terms, $f(x)$ is a linear decision boundary for its input vectors. However, if the training data is not linearly separable, then this algorithm is not guaranteed to converge.

The easiest and most famous example of not linearly separable function is XOR [41], which is defined in the following table:

[4] https://en.wikipedia.org/wiki/Affine_transformation

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

If we plot XOR, we will notice that there is no such a straight line can be drawn to perfectly separate two classes of dots. Therefore, a non-linear decision boundary is needed, which is achievable by introducing non-linearity to the neuron.

2.3.2 Non-linearity

The non-linearity is introduced by making the output of each neuron non-linear. This requires non-linear functions. These non-linear functions are ofter referred to as *activation functions*. However, there are certain traits of a non-linear activation function that are preferable when it comes to the training of a neural network. For instance, a non-linear function is preferred when it is continuous and differentiable [42].

The most widely used non-linear functions are hyperbolic tangent function (i.e., tanh), rectified linear unit (i.e., ReLU [43] and sigmoid function (i.e., logistic function)). The mathematic forms of the functions are the following:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$ReLU(x) = \begin{cases} x & x > 0 \\ 0 & otherwise \end{cases}$$

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Modern neural networks tend to have many layers. ResNet is proposed by K. He et al [44] that has a depth of up to 152 layers. This is eight times deeper than VGG nets [45]. Certain properties of non-linear functions such sigmoid and hyperbolic tangent function make it impossible to be applied in modern neural network training. The numeric stability of ReLU makes it the dominant choice nowadays.

2.3.3 Loss functions

Neural network can be applied as a supervised learning algorithm. The supervised training process requires both input data, denoted as X along with the ground truth (i.e., labels) that is denoted as y. The output of the neural network will be compared to the ground truth.

In machine learning, it is preferred to know how wrong an algorithm is instead of measuring how correct it is. Even though these two are usually mutually derivable. How "wrong" the output is is calculated by a certain loss function or cost function, which is the target function to be minimized.

$$L(\theta) = L(y, \hat{y}; \theta)$$

θ represents the parameters of target function, y is the true label and $\hat{y}$ is the output. Mean Squared Error (abbr. MSE) and Cross Entropy (abbr. CE) are the most widely used loss functions.

Mean Squared Error

Mean Squared Error or quadratic loss function, is widely used in regression problems. MSE is defined as the following:

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Cross Entropy Loss

Cross Entropy Loss is commonly used in classification problems, which is computed as the following:

$$CE(y, \hat{y}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i$$

In the context of information theory, cross entropy describes the difference between two probability distributions. When solving a classification problem, the output of the neural network is a probability distribution with regard to all the classes presented in the training data. Each component within this distribution represents the confidence level of the neural network regarding the corresponding class. The goal of training a neural network classifier is to make the model as confident as possible for the true labels.

2.3.4 Backpropagation

The training of a neural network is usually a feed forward process, meaning the computation flow goes from input to output, one end to another, as shown in figure 2. There is no loop inside the architecture. When a feedback loop is formed, the neural network architecture is often referred to as Recurrent Neural Network [42].

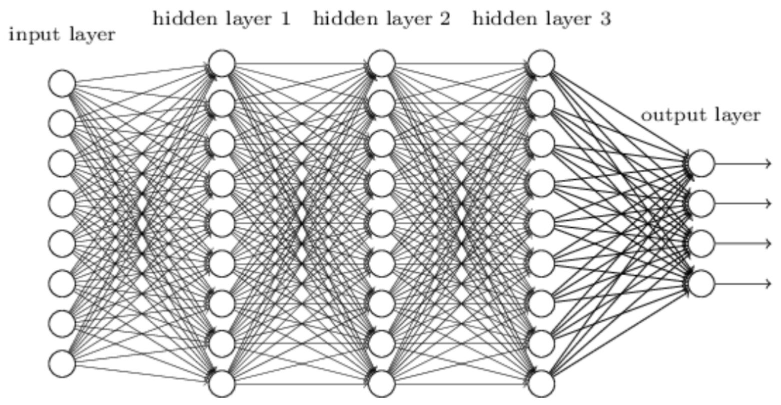


Figure 2: The Architecture of a fully-connected feedforward network.

Intrinsically, the training of neural networks aims to find a set of neuron weights so that the corresponding loss functions is minimized. The loss, i.e., the error that describes how bad the network is doing, will be computed once the control flow reaches the end of the flow. An effective tool to numerically reflect how much the weights should be adjusted is backpropagation.

As a short form of ”backward propagation of error”, backpropagation was invented in 1970s. It was until 1986 when Rumelhart, Hinton, and Williams [46] pointed out its importance with regard to the training of neural networks, i.e., updating the weights of each neuron in a network. In the paper, D. Rumelhart et al. described how backpropagation could be applied to train neural networks not only more effectively but also more efficiently. The application of backpropagation made neural nets useful for solving previously insoluble problems. The backpropagation algorithm has been the core of training in neural networks nowadays.

The feed forward process and backpropagation combined can be concluded as the following:

1. Take the input and go through the network until the output layer;
2. Compute the loss based on the loss function;
3. From the output layer backwards, adjust the weights of neurons on each intermediate layers until the first layer;

The effectiveness of backpropagation has been proven by many studies and experiments [47, 48, 49, 50]. There are two main handy tools available, partial derivative and chain rule in calculus to enable backpropagation

- partial derivative: for a multivariable function $z = f(x_1, x_2, ..., x_N)$, the partial derivative of each component, is denoted as:

$$\frac{\partial z}{\partial x_i} = \frac{\partial f(x_1, x_2, ..., x_N)}{\partial x_i},\ i = 1, 2, \cdots, N$$

- chain rule: define $z = f(g(x))$, the derivative of z with regard to x is:

$$\frac{dz}{dx} = f'(g(x)) \cdot g'(x)$$

These two tools are essential to the training neural network, specifically to backpropagation algorithm. The way neural networks work is simple. Each neuron does a even simpler job. It takes the output of the neurons from last layer and does an affine transformation as discussed previously. The value then is transformed by a non-linear function. The transformed the value simply becomes the input of next layer. This makes a neural network a giant composite function with all the weights and biases of its neurons as its variables. Partial derivative is needed to handle multivariate functions.

Chain rule is crucial because of the nature of Neural Networks. Since a multi-layer neural network is virtually a non-linear function, a one layer neural network with non-linear activation approximates a non-linear function too. Thus, a multi-layer network is chained by many non-linear functions, i.e., each layer of the network. The earliest layer, as a non-linear function, is embedded the deepest, as shown in the formula:

$$f_n(f_{n-1}(f_{n-2}(\cdots f_1(x))))$$

with f_i denotes the function that layer i computes. Clearly, the change of weights on early layers tend to have more impact than later layers. To propagate loss to the earliest layers, chain rule is essential since it take all the adjustment that has been made on later layers and provide that information to early layers.

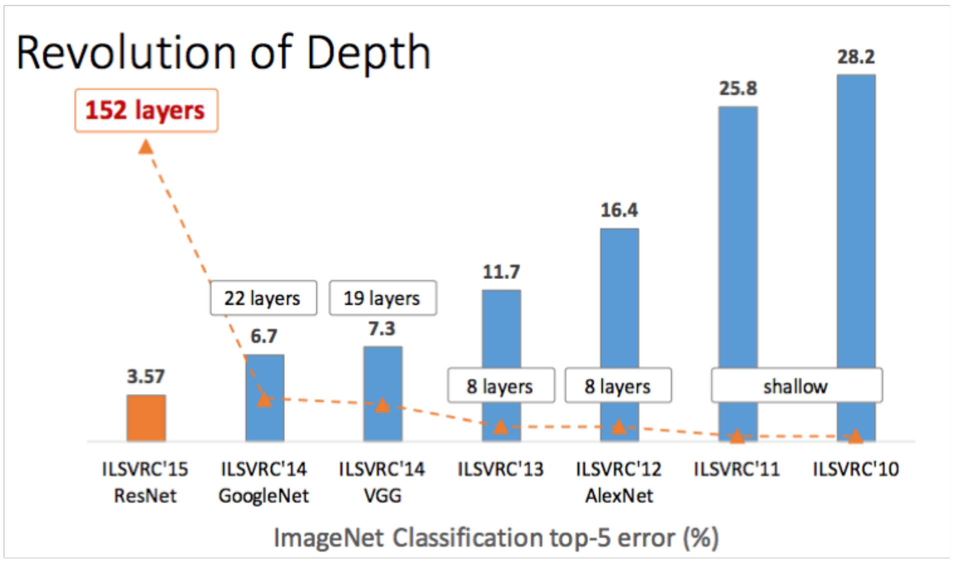


Figure 3: A plot shows the revolution of depth.

Deeper, the better?

As discussed previously, more layers simply mean a bigger composite function. There is a reason for the emerging of bigger and deeper models, as can be seen from figure 3 [5]. Apparently, deeper and bigger models tend to perform better. Deeper models tend to have more variables, which make the model more capable of approximating more complicated functions. An appropriate analogy is that, a system provides more knobs available for people to turn, which means the system is capable of performing more sophisticated tasks.

In this thesis work, the aim is not to compare deep and shallow models to see which performs better. The goal is to investigate if a neural network can

[5] https://medium.com/@Lidinwise/the-revolution-of-depth-facf174924f5

be trained to traverse graphs and perform path finding tasks. Therefore, the focus will not be to find the best network architecture or to achieve as high accuracy as possible. As long as the model perform reasonably well, the network topology will be used.

2.4 Data Augmentation

It is commonly known that machine learning algorithms tend to have better performance with access to more data under the condition that data does not suffer from bad quality. Clearly, the training size required depends on the complexity of the algorithm. It is also believed that the training data required to train a Neural Network model should be as large as possible. However, deep learning practitioners tend to argue that if the data quality can be guaranteed, thousands of training samples can already be used to train a fairly good model. As can be also seen from figure 4, large deep neural networks will outperform small ones with a large margin but on the condition that the amount of labeled data is really large [51]. When training size is small, there is no obvious advantage of using neural networks.

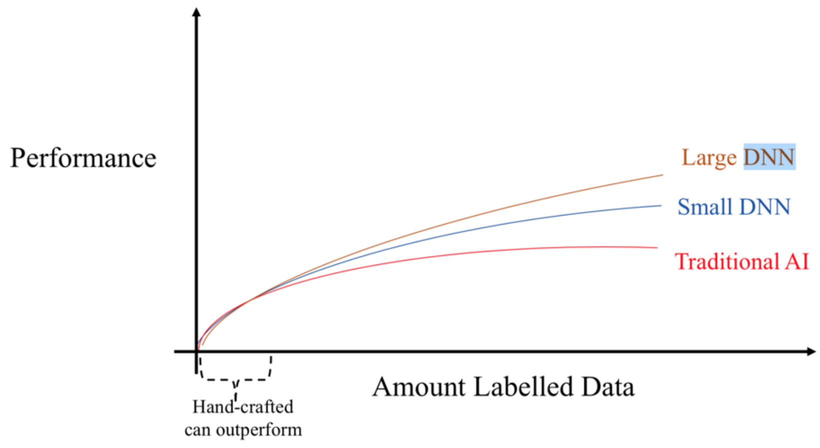


Figure 4: A comparison of traditional machine learning algorithms and Deep models.

Data augmentation has been widely used in many machine learning tasks such as image, video and audio classification as an effective method to enlarge the training set. To create new samples from the original training data for image classification problems, techniques such as flipping, distorting, adding noise, cropping are commonly used techniques [52]. Similarly, noise, distortion can also be added to video and audio [53, 54] for the purpose of increasing the size of training data. Manually labeling data is time-consuming and error-prone. Therefore, algorithmic data augmentation methods have been widely used [55, 56].

The reason that these techniques are effective on images, videos (i.e, sequence of images) and audios is that these types of data have certain spatial traits [57]. Stated differently, one single pixel in an image or one intensity value in an audio file does not convey much information. It is the combination of thousands of pixels or intensity values that is meaningful. Moreover, pixels or intensity values that are spatially closer together are more closely related than those that are far away. Local features matter.

In this thesis work, data augmentation is also applied since as will be shown in section 4, the training samples show certain spatial features that can be utilized to augment the dataset. Since we are mainly using simulation data, this means we can generate as much data as we wish. Data augmentation does not seem necessary. However, in reality, the size of the real dataset is limited, it is then useful to apply data augmentation so that neural networks can be better trained.

2.5 Design Choices

2.5.1 Neural network architecture

This thesis started with studying how route planning in road networks is typically done and how neural networks can be applied for graph traversal and path finding. It turned out that there was little we could build upon except the work done by Alex et al. [24] and Y. Xu et al [25]. Given all the encouraging results from these two work, we did not build our work on them for the following reasons:

1. We estimated that it would take significant amount of time to implement DNC or PCNN even though there is source code available for both architectures [6][7]. As a proof of concept for our purpose, spending too

[6] https://github.com/deepmind/dnc [24]
[7] https://github.com/pcnn/traffic-sign-recognition

much time on complicated Neural Network architecture is not pragmatic.

2. it takes little time to implement a mulit-layer perceptron and verify its feasibility. If it works, there is no urge to adapt to more complex models. If a simple model fails to work, we still can choose complex models.

Given that our model is yielding sequence of edges or road segments, it is natural to think that Recurrent Neural Network (RNN) [58] should be the first choice. The reason not to start with it is similar as argued previously. Furthermore, the over-engineered nature of RNN makes training and inference computationally expensive. Therefore, we were motivated by starting with simple architectures.

2.5.2 Regression or Classification

When applying supervised learning algorithms, people first need to define the problems as either regression or classification. As discussed previously, we use the paths generated by shortest path algorithms as our training labels. Shortest path algorithms aim to find paths that minimize the total edge weights. It is reasonable to train neural networks aiming to minimize sum of edge weights.

- **regression**: the target is to minimize sum of weights directly; therefore, mean squared error is opted;
- **classification**: cross entropy loss is usually preferred in this case; the model performance is indicated by the accuracy.

As will be shown shortly, the problem is solved as classification for the following reasons:

1. **interpretability**
   Mean squared error outputs a real number. In our case, it is not self-evident based on the output how a model is performing. Whereas for a classification problem, a direct indicator is accuracy, which is directly interpretable. As will be shown, it takes more engineering effort to generate paths if we go with regression. From a programmer's perspective, the output of the neural network model immediately indicates which edge should be picked when it is classification. It is not the case if the problem is treated as regression.

2. **consistency**
   Initial step taken did show that regression was feasible. However, the later steps showed that classification was a better choice. Therefore, to reduce the engineering work and make the work more consistent, we treated it as a classification problem.

2.5.3 Exploration & exploitation

The exploration and exploration strategy has been applied in various fields such as sociology, biology, business development, organization science [59], search and optimization algorithms [60, 61]. The classic definitions of exploration and exploitation given by James G. March [59] is that exploration focuses on new possibilities whereas exploitation focuses on old certainties. Exploitation means following the rules and avoid risks and exploration embraces risks and uncertainties.

There is an essential trade-off between exploration and exploitation. There has been much debate and research regarding this trade-off in many domains [62, 63, 64, 61, 60, 59]. It is reasonable to be consistent and follow the rules, meaning exploit all the time. However, this bias causes problems. A simple example is building a recommendation system. One trade-off is whether the algorithm should recommend contents fully based on customers' history. The answer is no. If users always receive very similar contents, the collected data will be less and less diverse. This will eventually drain the algorithm. To mitigate this, exploration techniques are introduced. Recommendation systems can push new contents tentatively. The users' interests can be further explored and user data can be enriched.

There are similar examples from enterprise development. It is necessary that a company work on its core product, which is exploitation. However, once a company passes a certain stage of its growth, it becomes dangerous to invest too much on the old product. It is necessary to explore new products to continue the growth, which is exploration. There is an analogy in biology. As pointed out by R. Dawkins [65], in order to for a gene to survive, a gene has to exploit by reproducing and explore by mutating.

In the study, it is interesting to apply this strategy. Human driving behavior has randomness that is shaped by environment. Deterministic computing models such as neural network fails to capture the randomness. Neural networks exploit the graph by deterministic inference. As developers, we need to help the models to explore by randomizing the output with a certain probability (e.g., 10%) and see how that affects the paths.

2.5.4 Path Similarity

Path similarity can be categorized as trajectory similarity measures. Measuring trajectory similarity has many important use cases in various fields [66, 67, 68]. There are many standard methods.

1. Hausdorff distance; it is defined as the greatest distance of all possible distances calculated from a point in one set to the closet point in the other set [69]. We define the two point sets as A and B and d is some distance function between two points. The mathematical formula is the following:

$$distance(A, B) = \max(\max_{a \in A} \min_{b \in B} d(a, b), \max_{b \in B} \min_{a \in A} d(b, a))$$

   There are cases where trajectories are very different but can have small Hausdorff distance, as shown in figure 5, which is due to the fact that Hausdorff distance does not take continuity of paths into consideration.

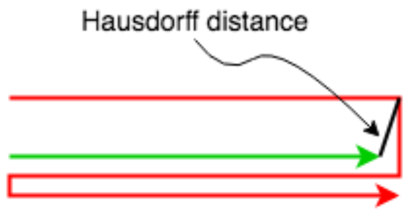


Figure 5: A picture shows an edge case where two very different paths have small Hausdorff distance

2. Fréchet distance; As shown in figure 6, all points a in curve A is mapped to the points in curve B through continuous function $\mu : A \to B$. Then, the distance between curve A, B is:

$$\max_{a \in A} d(a, \mu(a))$$

The actual Fréchet distance calculation needs to find the map that minimize the above distance. It can be denoted as the following:

$$Frchet - distance(A, B) = \min_{\mu} \max_{a \in A} d(a, \mu(a))$$

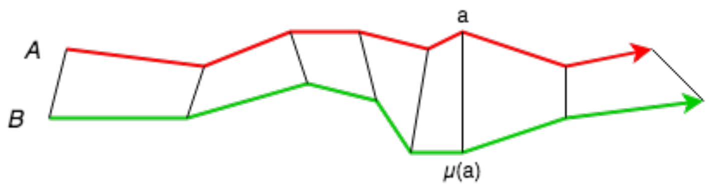


Figure 6: A picture visualizes the calculation of Fréchet distance

3. Edit distance; Edit distance is self-explanatory. The edit distance is calculated by how much one path has to be changed to be another path [70]. A simple example is shown in figure 7. In order for curve A to be curve B, two segments of A need to be edited. Therefore, the edit distance between A and B is 2.

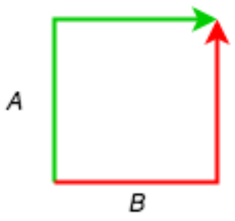


Figure 7: A picture is used to explain Edit distance with the grid hidden.

3 Methodology

The research question of this study is, *Are neural networks applicable to real-time route planning tasks in a road network?*. To answer this question, two graph traversal strategies are proposed; random graphs are generated using customized procedures; neural network models are trained so that they can conduct pathfinding tasks in random graphs. Therefore, this section consists of four parts. Firstly, we give an argument about the chosen research method and compare it with other research methods. Secondly, we briefly introduce how these graph traversal strategies are designed. Thirdly, the evaluation metrics are discussed. Lastly, the analytic methods and experimental settings will be provided.

3.1 Research Methods

As discussed previously, this study involves both quantitative and qualitative evaluation. When it comes to a broader view, a few high-level research methods are available such as *experimental*, *descriptive*, *fundamental*, *applied*, and *empirical research* [29]. The main focus of this thesis is not to develop new theories or concepts regarding neural networks or shortest path algorithms. Therefore, *conceptual* and *fundamental research* are not applicable in this case. The *applied research* method is often used to answer specific questions or to solve known and practical problems. Since route planning is a well-known and practical problem, this research method could be adopted. However, since the research question does not aim to build a useful product (e.g., deep learning-based pathfinding software package), the *applied research* method is not used in this study.

A deeper understanding of the research question shows that *empirical research* method is the most applicable one. The empirical research involves evidence, observations or experiences from experiments [29]. It focuses on real people and situations and derives knowledge from tests. This study aims to apply neural network models to real-time route planning. Meanwhile, the effectiveness of a Machine Learning model can be evaluated quantitatively and qualitatively. As for the data collection methods, *experiments* [29] are chosen given that experiments should be designed to draw conclusions. Therefore, other methods such as *Case Study* and *Interviews* [29] are inapplicable. The performance of the model will be measured by the proposed evaluation metrics, and the observations collected by experiments will be statistically analyzed. Then an answer can be provided to the research question.

3.2 Graph Traversal Strategies

This study aims to train neural network models to perform pathfinding tasks in road networks. To achieve this, the model needs to process the graph information, make predictions, and yield paths. There are different ways to process graph information. As discussed previously, Xu et al. [25] used D-S evidence theory to fuse road segment length, path saturation [26], and vehicle speed data. Zhang et al. [28] used matrices to represent graph information. Their methods both introduce the overhead of matrix operations and are limited to a small number of graph properties. There should be an alternative way to represent node and edge information. Therefore, we first introduce greedy strategy. As a proof of concept, the model is only provided with edge weights and it is trained to pick the edge with the smallest weight (section 4.2). Then we designed shortest path strategy where more node and edge properties are encoded and provided to the model. The model is trained to be smart about its predictions (section 4.3). Finally, the trained model is applied to a road network to qualitatively analyse the model performance. More details can be found in section 4. In the next section, we introduce how the model is evaluated.

3.3 Evaluation Metrics

The focus of this study is pathfinding using neural network model. We need methods to evaluate both the effectiveness of the model and the quality of generated paths. In graph theories, shortest path algorithms are preferable to generate the benchmark. As discussed previously, Hausdorff and Fréchet distance both give real numbers as results. A distance of zero apparently means two paths are identical. However, there is no intuitive way to immediately tell how different two paths really are. Edit distance is not optimal neither since:1) it is difficult to first define how to "edit" distance; 2) two paths can be very close yet completely different (e.g., two parallel highways). In this case, a path generated by the model can still be very good but classified as a "bad" path. Therefore, the above three metrics will not be considered in this study.

We used cross entropy loss to train the model, therefore, one can argue that model can also be evaluated with accuracy. The model accuracy in this work is interpreted as how likely a model makes the correct prediction on the next node. For greedy strategy, this metric is enough since it only cares about the next node. Higher accuracy means the model is learning this strategy better. However, for shortest path strategy, being able to correctly infer the next node does not directly mean that the model is learning the strategy well. Therefore, we will propose our own evaluation metrics. When it comes to comparing the neural network-generated path with the optimal path, two aspects are of great interest:

1. Effectiveness: can the model find a path? What is the probability that the model fails to find a path from A to B in a graph?
2. Path quality: are the paths generated by the model good? The path can have either fewer steps or smaller total edge weights.

3.3.1 Destination arrival rate

The effectiveness of a model indicates whether it can do its job. After the model is trained on a graph or a road network, it is interesting to know whether it can work. The method is direct: sample a pair of nodes and ask the model to give a path between them. If the model has a high chance to give a path to the queries, we can treat it as a functional model. The *arrival rate* can be defined as the following:

$$\textit{arrival rate} = \frac{\textit{number of pairs given a path}}{\textit{number of sampled node pairs}}$$

Higher arrival rate means higher model effectiveness.

3.3.2 Path weight sum

People care about how fast they can reach their destinations. It is meaningful to know if the paths found by the model are good-enough, i.e., the path quality. Edge weight can be treated as the time spent to travel along a road. If we define the shortest path (i.e., benchmark) as $path_{shortest}$ and the path taken by the model as $path_{model}$. The *time efficiency* of the model is:

$$\textit{time efficiency} = \frac{\sum_{weight} path_{shortest}}{\sum_{weight} path_{model}}$$

Smaller total edge weights indicate better path quality.

3.3.3 Path edge count

It might be annoying for a driver to frequently change to different roads when traveling. A lot of people would prefer traveling as less road as possible to reach their destinations. Therefore, it makes sense to evaluate how many edges the model has to take to find the destination. We can define *edge efficiency* as the following:

$$\textit{edge efficiency} = \frac{\textit{number of edges of } path_{shortest}}{\textit{number of edges of } path_{model}}$$

3.4 Data Collection

Once the model is trained on data generated from a random graph, we can do experiments to test its generalizability, arrival rate, and path quality. Using the same procedure, more new graphs are generated to see if the trained model can perform well; the evaluation metrics here is classification accuracy. The collected observations will be provided with scatter plots.

The trained model will be applied to generate paths in new graphs given pairs of origins and destinations. Those paths will be compared with the corresponding shortest paths. Meanwhile, as discussed previously, the exploitation and exploration trade-off is adopted; different extent of randomness (e.g., 0, 0.25, 0.5, 0.75) will be experimented and analysed with scatter plots and boxplots. The real-time pathfinding performance will evaluated with the same procedure except that the edge weights are modified randomly before the model makes predictions. Statistics regarding arrival rate, time efficiency, edge efficiency will be provided. As for the model effectiveness, ideally, we want to train the model so that it can always find path between a pair of origin and destination (given there is one). For path quality, we want the paths generated by the model be close to the shortest path.

3.5 Experimental Settings

The hardware used for the experiments is shown in table 1. The hardware is host on Azure Virtual Machine [8]. The main programming languages used in this work is Python 2.7 [9] and Julia 0.6 [10]. The software packages used during the development of the prototype and the experiments are listed in appendix B. All experiments were conducted on the same machine and all source code is available from GitHub [11]

Item	Specification
Operating System	Ubuntu 16.04.3 LTS
CPU	Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz
GPU	NVIDIA Tesla K80
Memory	56 GB

Table 1: A table shows the hardware used during the experiments

[8] https://azure.microsoft.com/en-us/services/virtual-machines/
[9] https://www.python.org/download/releases/2.7/
[10] https://julialang.org/downloads/
[11] https://github.com/zhoutianyu16tue/PathFinder

4 Algorithm Implementation

4.1 Dataset

4.1.1 Random graph generation

Real road networks are complex and hard to process. A commonly used open source map tool is OpenStreetMap [12]. The map data has much information such as residential area, building, pedestrian, coordinates. This would only slow down our initial steps by having too much temporarily unnecessary information. It is important to mathematically abstract the key information. Therefore, in this study, we designed our random graph generation procedures to support the initial investigation of greedy strategy and shortest path strategy.

Greedy strategy

The random graph is generated with some practical constraints such as each node has maximum 4 out neighbors and the edge weights follow a uniform distribution of $[0, 1]$ for simplicity. A example of such graph can be seen from figure 8. The graph can be generated as either directed or undirected.

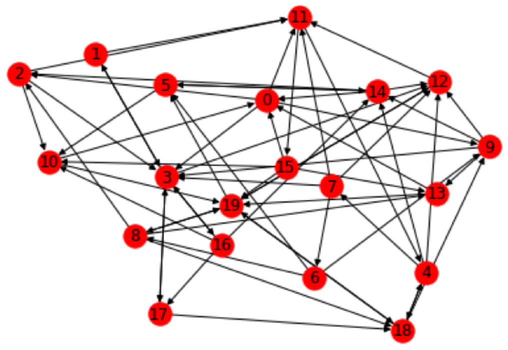


Figure 8: A random graph has 20 nodes.

Shortest path strategy

[12] http://openstreetmap.org

For the greedy strategy, the generated random graphs have no geographic information provided for the model and indeed there is no need for such information. To better approximate a road network and to better guide the search of the neural network model, certain assumptions must be made to the graph.

- Nodes are randomly given (x, y) Euclidean coordinates.
- Nodes are more likely to be connected when they are geographically closer.
- Each node has maximum 4 and minimum 2 out neighbors;
- Weights of edges are the multiplication of nodes Euclidean distance and a random relaxation factor.

The first two assumptions are very intuitive because the nodes that are geographically far away tend to use other intermediate nodes as hops to reach each other. For instance, there is little chance that there will be a direct path from Kista to Stockholm city center since Solna always serves as a transportation hub. For the third assumption, a drive has limited number of choices to when standing at a crossroad, as shown in figure 9. More complex cases such as roundabout might emerge. It is not much more complicated and can be adapted correctly.

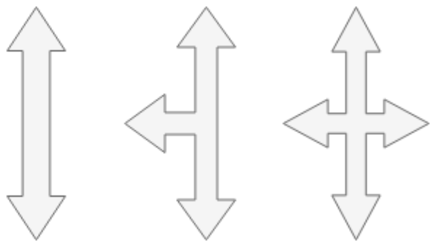
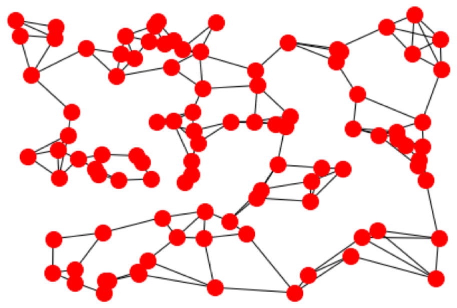


Figure 9: A example shows the out degrees of a crossroad in a road network.

The last assumption might be confusing at the first glance. Firstly, it usually takes longer to travel through a long road segment. However, if such road has higher speed limit, it might not be the case. In urban or rural areas, the speed limit does not vary drastically. The relaxation factor gives the edge weights some flexibility In this work, the relaxation factor is randomly sampled from a uniform distribution, $[0.5, 1]$ for example. If we want the edge weights to be more flexible, we can simply widen the range of relaxation factor to $[0.1, 1]$ for instance.


(a) A random graph has 100 nodes.

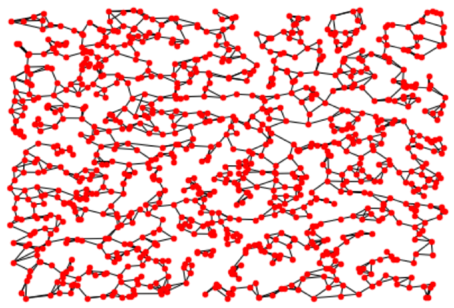

(b) A random graph has 1000 nodes.

Figure 10: Two examples show the output of random graph generation procedure

Two graphs can be seen from figure 10a and figure 10b. An observation is that the random graph with 1000 nodes presents certain small-world traits [71]. Stated differently, if we treat this graph as the road map of a city or a province, there are communities (e.g., different zones of a city or different cities of a province) of nodes clustering together. These communities are connected with each other by a few "important" edges. Those important edges are main roads of a city and highways between cities. As will be discussed shortly, the importance of roads can be described with centrality coefficients.

4.1.2 OpenStreetMap

OpenStreetMap (OSM) is a free editable world map that is collaborated by many people [13]. People can edit and download the map data from the website, as shown in figure 11. A screenshot of northern part of Stockholm is shown in figure 11 [14]. This area has the bounding box of $min\_longitude = 17.7755, max\_longitude = 18.0464, min\_latitude = 59.3274, max\_latitude = 59.4421$.

[13] `https://en.wikipedia.org/wiki/OpenStreetMap`
[14] `https://www.openstreetmap.org`

Figure 11: A screenshot of northern Stockholm from OpenStreetMap.

OSM provides APIs for many programming languages. In this work, we mainly use the Julia OSM API [15]. With the API, the road network can be conveniently extracted from the map data, which is essentially a XML file.

4.1.3 Other options

The host company of this thesis work, LakeTide AB, is in possession of a dataset that reflects how people travel in Stockholm city. It would have been ideal if this dataset could be used. However, due to the sensitivity and possible privacy issues, this thesis does not use this dataset. There are also other map applications available for use. OpenStreetMap is chosen due to its flexible APIs, free downloading map, and detailed documentation.

[15] http://openstreetmapjl.readthedocs.io/en/stable/

4.2 Greedy strategy

Pathfinding essentially serves for humans. It is constructive to think what human instinct is when placed in an unfamiliar road network. Most people would try to be greedy when it comes to traversing a graph, meaning to just choose the road that seemingly has the smallest weight, i.e, time:

$$time = \frac{road\ length}{speed\ limit}.$$

Greedy strategy makes the locally optimal decision at each step. Lacking geographic information and global knowledge about the road network, it is natural to just pick the edge that takes the least time to travel along. As the first stab, we investigated whether a neural network could learn the greedy strategy. All code is available from github [16].

4.2.1 Algorithm

The sample size of a graph is equal to the number of nodes in the graph since for each node, the greedy choice is uniquely determined. Mathematically, the sample size complexity is $O(V)$ where V is graph node size. The input for the neural network during training and inference is generated for each node, which is simply a vector of edge weights of all out neighbor of the node. If a node has less than four out neighbors, then the vector is just padded with 1's. The model only infers the next node based on limited local information.

As shown from figure 12, $edge(1,3)$ has the smallest weight among all out edges of node 1 therefore this edge is chosen. The input sample is:

$$X = [0.4, 0.3, 0.5, 1.0], y = 1$$

The label simply indicates which edge should be chosen given this input vector. The index does not mean anything to the model. It only makes sense to us, as developers. The nodes inferred by the model is manually added, as show in the following pseudocode:

```
define greedy_path_finder(G, src, dst, model,
    invalid_path_threshold):
```

[16] https://github.com/zhoutianyu16tue

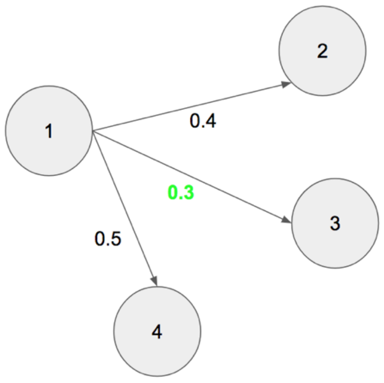


Figure 12: An example shows picking the edge with smallest weight for a node.

```
path = [src]
cur_node = src

while True:
   # The path is invalid after traversing certain number of nodes
   if # visited nodes >= invalid_path_threshold:
      return FAILURE

   next_node = nn_infer_next_node()
   # Add the new node to the list
   path.append(next_node)

   if next_node is dst:
      return SUCCESS

   # Update the current node
   cur_node = next_node
```

The algorithm that the neural network infers next node is as the following:

```
def nn_infer_next_node(G, cur_node, model):
   generate model input: network_input
   next_edge = model.predict(network_input)
   Return the next node according to next_edge
```

4.2.2 Discussion

A simple multi-layer perceptron was trained and it could learn this strategy well. However, it is evident that greedy strategy can get stuck in a loop easily. The aforementioned exploitation and exploration strategy can be applied to escape the loop. This might just lead to another loop. When a driver find himself traversing the old road, he can just pick a random road and restart. However this is not a good strategy by itself since only limited information is provided to the model. In real life, drivers can have more information such as main roads, directions, traffic conditions, and the approximate distance to the destinations. A more advanced strategy should be investigated.

4.3 Shortest path strategy

Human memory limitations prevent people from remembering the entire road network topology, traffic densities, and other details that optimal algorithms would ordinarily be privy to. Even so, certain knowledge of a graph should be encoded to guide to model to learn. The knowledge regarding a graph belongs to two main categories, local knowledge and global knowledge.

4.3.1 Local knowledge

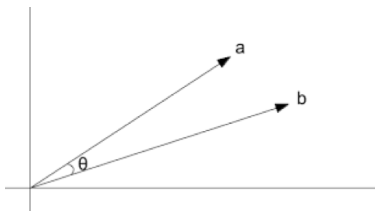
Edge weights are essential for a neural network to learn shortest path strategy. Edge weights alone do not effectively guide the search of a neural network model. As a second attempt, edge centrality is included. Centrality indicates how "important" a node or an edge is in a graph. In the context of social network analysis, the "importance" describes the most influential person(s). In this study, edge importance is reflected through shortest path algorithms, i.e., shortest path betweenness centrality [72]. If an edge is frequently included when finding the shortest paths between nodes, then the edge has higher centrality value. Centrality somehow reflects how humans act about what road to pick. Popular roads are more likely to be chosen. As will discussed shortly, popular roads attract more vehicles and hence more traffic congestions, which in return affect human driving behavior.

4.3.2 Global knowledge

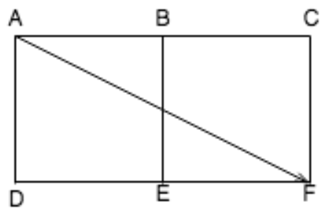
Two main global heuristics are provided to the neural network, geographic distance (i.e., euclidean distance) and cosine distance. These two indicators reflect the two questions that a human drive would frequently ask:

- Am I getting close?
- Am I in the right direction?

The Euclidean distance gives the model the information about how far it is away from the destination. If the next step is taking me further away from destination, a human driver would probably feel reluctant to pick that edge. This might not always be the case since the edge can be a highway with much higher speed limit therefore takes less time to drive through.



(a) An example illustrates cosine distance.



(b) A example shows which edge to choose based on cosine distance.

Figure 13: Two examples show the concept and effect of cosine distance respectively.

Cosine distance is guiding the model to approach the destination in the right direction. The geometric representation of cosine distance is shown in figure 13a. The mathematical form is the following:

$$\cos(\theta) = \frac{\overrightarrow{a} \cdot \overrightarrow{b}}{|\overrightarrow{a}| \cdot |\overrightarrow{b}|}$$

Figure 13b gives an example of the effect of cosine distance. If we try to move from point A to F, the first step is to either move to point B or point D. Since $\angle BAF$ is smaller than $\angle DAF$, we are probably going to choose point B.

Most human drivers simply follow the road segments that points them to the destinations. However, different situations give rise to various choices. For example, there is a highway that is a detour (i.e. leading to a slightly wrong direction at the beginning but eventually pointing to the destination) yet takes significantly less time because of much higher speed limit. There is no easy answer to whether a driver take this road in this case. To learn the shortest path strategy on a simple graph, we limit ourselves in a small set of factors that might affect drivers. More heuristics will be introduced when it comes to real road network learning.

4.3.3 Ground truth

Ideally, everyone gets to travel along the optimal paths, However, this rarely happens in real world. Finding an optimal path for one person might affect the choices of another because of the dynamics of traffic. It is suspicious whether it is meaningful or even correct to use the paths generated by shortest path algorithms as the ground truth. Given that A* search extends Dijkstra's algorithm, it is able to take into account some customized heuristics. As discussed previously, heuristics based search does not necessarily find the best solution but a good enough solution yet much faster. We realized that there are certain constraints [73] must be satisfied in order to provide good heuristics to A* algorithm. Still, we used Dijkstra's algorithm to compute the paths and use them as ground truth for the neural network. It must be emphasized that the paths generated by the model are not necessarily the optimal ones. The aim to is to train the model to make locally optimal choices so the time complexity is constrained by $\Theta(E)$ and guarantee the path quality.

4.3.4 Algorithm

The implementation of this strategy is similar to the greedy strategy. One major difference is that each edge has more information such as edge centrality, euclidean distance, cosine distance, as shown in figure 14. Another important difference is how the data is sampled from the graph. Since the ground truth comes from shortest path algorithm, node pairs are randomly sampled from the graph and the shortest paths are generated from these node pairs. The edge of each path is encoded accordingly as one sample. Therefore, the sample size complexity is $\Theta(V^3)$ where V is graph node size.

The feature vector of each edge can be arbitrarily large. The input to the neural network is the concatenation of all these feature vectors as the following:

$$X = [0.1, 0.33, 0.41, \mathbf{0.3, 0.51, 0.11}, 0.5, 0.11, 0.98, \mathbf{0.0, 0.0, 0.0}]$$

Since we assumed that each node has maximum four out neighbors, the length of the input vector is $4 \times$ *length of edge feature vector*. The missing edge is replaced with a zero vector. In the case shown in figure 14, $Edge(1, 3)$ is chosen by Dijkstra's algorithm. Therefore, the label of this input sample is 1, indicating that the position of the edge that is chosen, shown in bold from above. As the implementation of greedy strategy, the model only infers the next node based on all local and global information it has. This strategy is essentially an extension of greedy strategy since it makes the optimal choice locally according to the current situation.

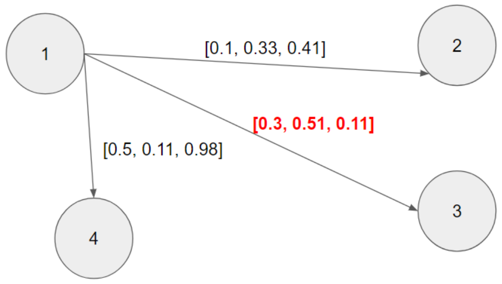


Figure 14: An example shows picking the edge according to A* search

4.3.5 Data augmentation

Data augmentation expands the training data and it is applicable when the original data set presents certain spatial traits. When it comes to greedy strategy, the only thing that the model needs to learn is that the edge with smallest weight should be picked. As for at which position the smallest weights are, the model does not differentiate. Stated differently, the input vector can be shuffled as long as the label indicates the position where the smallest weight is.

If we use the edges in figure 12 as an example, the input sample

$$X = [0.4, 0.3, 0.5, 1.0], y = 1$$

is conceptually identical to

$$X = [0.3, 0.4, 0.5, 1.0], y = 0$$

However, by exchanging the first two values, we can have a new training sample. Therefore, the dataset can be augmented with this technique for 10 or even 20 times. It is not only applicable for greedy strategy but also shortest path strategy. The order of the feature vectors shown in figure 14 can be arranged arbitrarily, as long as the label indicates which edge should be picked.

4.4 Road Network Learning

Road networks can be reasonably abstracted to weighted directed graphs. However, mathematical graphs do not usually include geographic coordinates of nodes and edges, which are the main components of road networks. Each node in a road network at least has longitude and latitude, in some cases with altitude. With the addition of geographic information, road networks tend to present a more clear mathematical topological structure. These features lead to the first two practical assumptions, closer nodes tend to connect and node degrees are usually small. Therefore, road networks are not conceptually more difficult than the random graphs as discussed in section 4.1.1.

Processing real road networks has certain practical difficulties. The major one is the size of the resulting graph. As discussed previously, the training sample size that can be generated from graph is $O(V^3)$ (with V the number of nodes). For a graph of 1000 nodes, the sample size is both too large and can not be generated in reasonable time. A small road network shown in figure 11 can easily have over 20,000 nodes even processed with medium granularity. It is intractable to train the model on the entire road network. Additionally, training on the complete graph does not scale when the graph grows even bigger. The solution is to randomly sample pairs of nodes and generate the training data. This not only reduces the time complexity of generating data but also allows the model to learn to generalize when encountering unseen nodes or edges.

4.4.1 Road network processing

The road network that can be abstracted from figure 11 is shown in figure 15. It is worth noticing that the granularity of a road network that can be extracted from OpenStreetMap can be configured manually [17]. Stated specifically, the density of the road network can be configured to exclude information such as *living_street*, *cycleway*, *bus_guideway*. In this way, we can control how fine-grained the road network is. As shown in figure 15, the density of the road segments is distributed unevenly across the landscape. As discussed in previous section, real road networks present small-world features, which is verified by this figure. Smaller communities appear in different regions on the map. Those small communities are connected by a few road segments. Some islands can also be seen from figure 15. These islands are seemingly isolated from the main land. However, as mentioned earlier, this is due to the exclusion of of vast number of residential road segments.

4.4.2 Knowledge for the model

As discussed in previous section, both local and global knowledge are provided for the model to learn the shortest path strategy. However, more knowledge can be provided to the model so that it better plans routes.

Firstly, traffic condition is an important factor that affects path finding. Paths that have less traffic are usually preferred. Traffic condition can be simplified as the number of cars on a particular road segment. More vehicles on a road has a better chance to cause congestion. However, the nature of the road such highway and speed limit should be considered. Highways, if not congested, usually have much higher throughput than urban roads. Meanwhile, a long road segment can afford to have more vehicles. Interestingly, traffic conditions can be either local or global knowledge. When facing an immediate choice of which road to pick, a driver tends to pick the one with less traffic. However, drivers receive information from radio or real-time navigation software that can help them avoid congestion. This makes traffic condition a global knowledge to the neural network.

Secondly, weather conditions should not be neglected and can be categorized

[17] http://openstreetmapjl.readthedocs.io/en/stable/routing.html

as global knowledge. Weather significantly affects traffic conditions [1]. As also pointed out by I. Tsapakis et al., travel time and speed are two major aspects that are greatly affected by weather. The investigation conducted by I. Tsapakis et al in the Greater London area gives concrete numbers showing how much travel time is increased by rain and snow, as show in table 2. Stated differently, weather has impact on mobility. According to LC Good-

weather	light	moderate	heavy
rain	0.1–2.1%	1.5–3.8%	4.0–6.0%
snow	5.5–7.6%	-	7.4 to 11.4%

Table 2: A table shows how weather increases travel time [1].

win [74], the speed on arterial routes is reduce by 10-25% on wet pavement and 30-40% with snowy or slushy pavement. Travel time delay on arterials can increase by 11-50%. The effect of weather can be treated as the relaxation factor that affects the road segment weights, as discussed in section 4.1.1. Thirdly, time of traveling affects drivers. Rush hours are different in cities and countries. Generally, there are morning and after-work rush hours. We believe that such aspects should also be considered in future work.

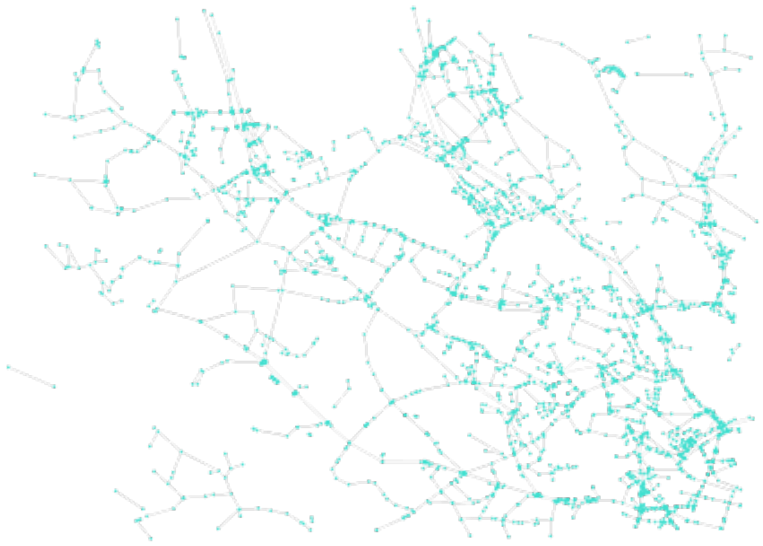


Figure 15: A plot illustrates the road network abstracted from figure 11.

5 Results

5.1 Greedy Strategy Learning

5.1.1 Training and inference

As discussed in section 4.2, each node can be represented as a 1-d vector, shown in figure 12. The length of the input vector is equal to the maximum out degree of the graph. In this case, it is 4. Therefore, for random graph with N nodes, the number of training samples that can be generated is N, meaning the input shape to the model is $(N, 4)$. A simple two-layer network, shown in table 3 is trained on a graph with 20 nodes. It is not surprising that the model can perfectly fit the training data given that the sample size is so small (i.e, 20, without data augmentation 4.3.5).

Input Layer
32-Fully Connected Layer
ReLU Activation
Softmax Output

Table 3: A table shows the architecture used to learn greedy strategy.

5.1.2 Model generalizability

It is worth concerning that the model simply memorizes the data without learning anything. Therefore, new graphs of different sizes are generated and tested to see how well the model performs on unseen graphs. Given that greedy strategy is fairly easy, the model should be able to learn and generalize well. To verify this, graphs with 20, 100, 500, and 1000 nodes are generated. Each size group has 100 new graphs. The model accuracy is averaged over each size group. The result is shown in figure 16.

As can be seen from figure 16, the model does not do well on unseen graph. This is expected since the training size is too small. The model has not been able to learn enough. A model with the same architecture is trained on a new graph with 100 nodes. The same test is conducted. The result is shown in figure 17. Apparently, the model trained with 100-node graph has better

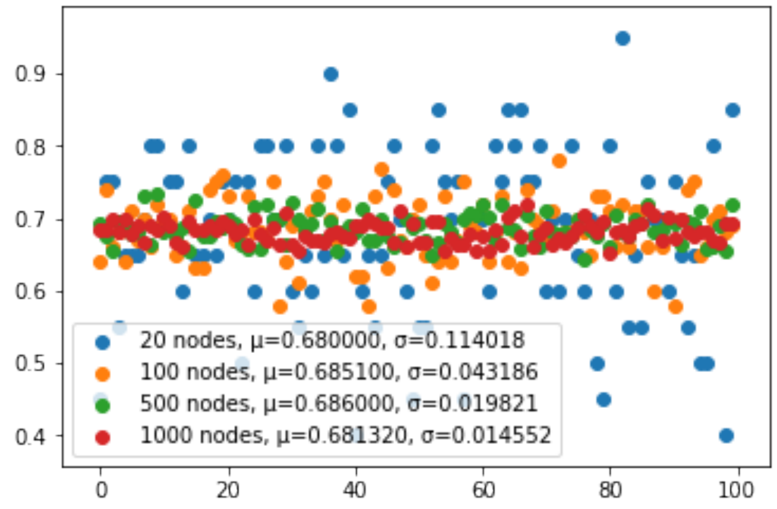


Figure 16: A plot illustrates the generalizability of a model trained on a graph of 20 nodes.

generalizability. The outcome is expected since now the model has learned from more information.

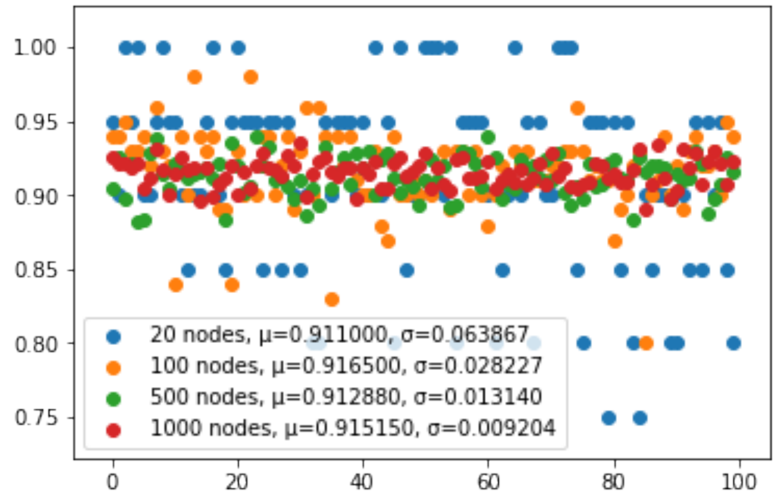


Figure 17: A plot illustrates the generalizability of a model trained on a graph of 100 nodes.

Data augmentation technique 4.3.5 is also applied and tested. A 100-node graph is augmented by 100 times, resulting in 10000 training samples. These samples were then used to train a model with the same architecture. The same test is conducted and the result is shown in figure 18. As can be seen, the model is able to generalize significantly better. This is because of the much larger number of training samples. The size the of the graph can indeed grow arbitrarily large. However, the model is able to learn the greedy strategy. For this work, one insight is that a model trained with more data tends to generalize better.

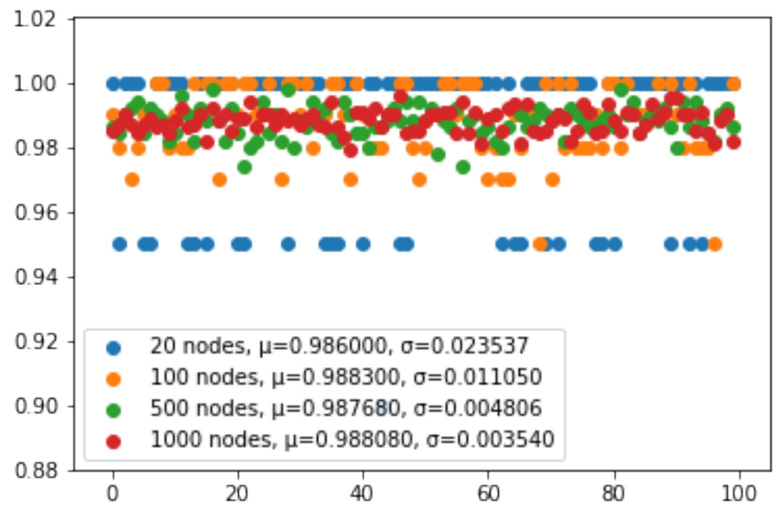


Figure 18: A plot illustrates the generalizability of a model trained on a graph of 100 nodes with data augmentation.

5.2 Shortest Path Learning

5.2.1 Neural network training

Similarly to the learning of greedy strategy, in shortest path strategy, each node can also be represented as a 1-d vector. The length of the input vector is decided by the number of edge features and the maximum out degree of the graph. When the neural network is standing at a node in a graph, it looks at all the out edges including the corresponding out neighbors and pick a node as the next step. The features considered by the model is shown in table 4. The (x, y) coordinates of the source and destination are also provided as a global knowledge (section 4.3) to the model. Given the constraints discussed

Feature	Description
Edge centrality	Analogous to the popularity of a road
Edge weight	Analogous to how long it take to traverse that edge
Direction	Cosine distance between the out neighbor and destination
Distance	Euclidean distance between out neighbor and destination
out neighbor location	The (x, y) coordinates of the out neighbor

Table 4: A table shows edge features used.

in section 4.1.1, the maximum out degree of the graph is 4. Therefore, the length of the input vector the neural network model is: $4 * 6 + 4 = 28$.

Since the effect of data augmentation has been validated in section 5.1.2, a simple multi-layer neural network with architecture shown in table 5 is trained on a random graph with 100 nodes and 2000 distinct node pairs are randomly selected. The training set is augmented by 10 times, resulting in 184760 training samples. To avoid overfitting, the training process is validate on the validation set, which consists 1890 samples generated from a new graph of 100 nodes. After convergence, the model has accuracy of 0.894 and 0.910 on training and validation set respectively.

The accuracy is not outstanding. However, accuracy is not the pursuit in this study. It is trivial to design a complex model and achieve high accuracy. A neural network that achieves high accuracy does not directly indicate that it can traverse graph effectively or efficiently. Having this accuracy means the model makes the wrong choice 10% of the time. This is not necessarily a bad thing since as discussed in section 2.5.3, the model should not only exploit the available information but also explore the graph with randomness, which is introduced by its mistakes.

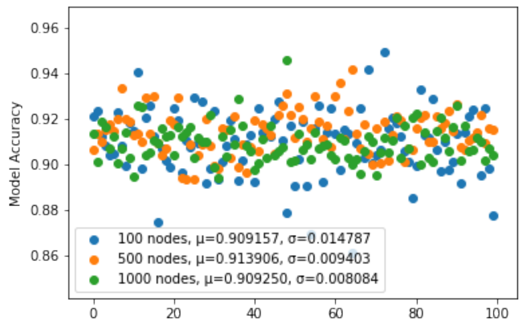
5.2.2 Model generalizability

It is also interesting to know if the model can generalize the learned strategy well to unseen graphs. Graphs with 100, 500, 1000 nodes are generated. Each size group has 100 new graphs. Each graph is used to sample 400 dis-

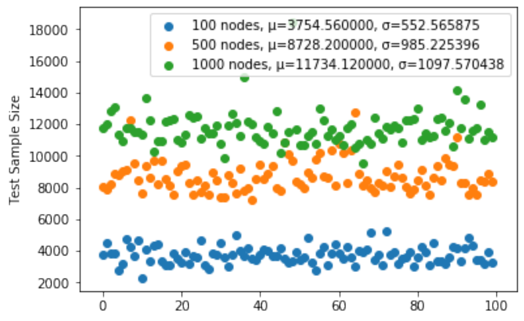
Input Layer
1024-Fully Connected Layer
ReLU Activation
Dropout (0.5)
1024-Fully Connected Layer
ReLU Activation
Dropout (0.5)
Softmax Output

Table 5: A table shows the architecture used to learn shortest path strategy.

tinct node pairs. The mean and standard deviation are calculated over each size group. The result is shown in figure 19.



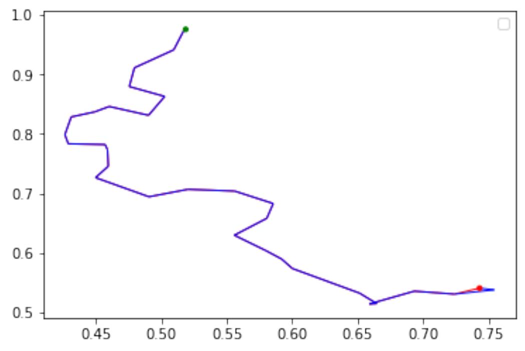
(a) A plot shows the model accuracy distribution on each size group.



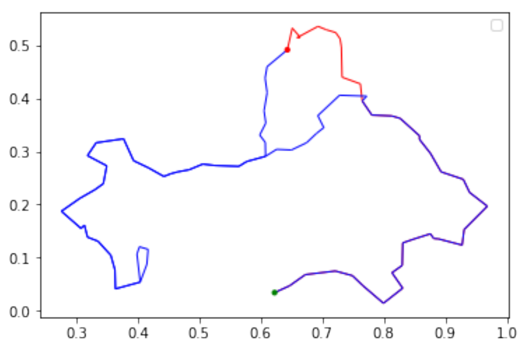
(b) A plot shows the sample size statistics over each run.

Figure 19: Two plots show the result of the shortest path strategy learning generalizability test.

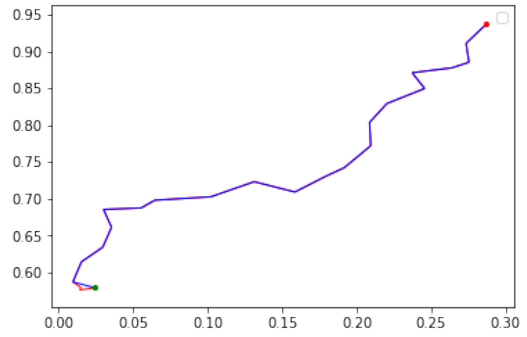
As can be seen from figure 19b, it is expected a bigger graph can generate more samples with the same number of node pairs. The average length of paths is linear to the size of a graph. As also shown in figure 19a, the model is able achieve the same level of accuracy with acceptable standard deviation on unseen graph. It can be expected that a model trained with higher accuracy can generalize even better. Since to achieve high accuracy with some model is not the goal of this thesis, the model optimization wil be left for future work.



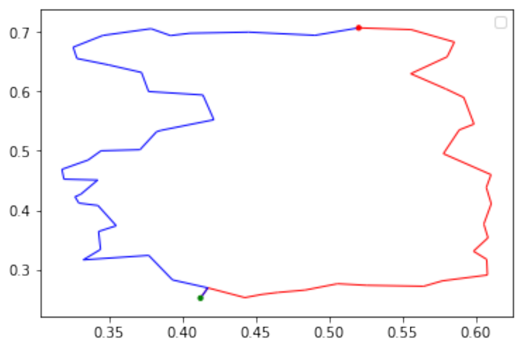
(a) A plot shows a good match between two paths.



(b) A plot shows the model struggled to find a path between two points.



(c) Yet another plot shows a good match between paths.



(d) A plot shows the model found a completely different path.

Figure 20: Plots show the comparison between model-generated paths and shortest paths.

5.2.3 Path visualization

The trained model is applied on the 1000-node shown in figure 10b. The paths generated by the model is compared with the corresponding optimal paths. The plots are shown in figure 20. Those four plots are selected carefully and therefore are representative.

Apparently, those paths are far from being optimal. One interesting observation is that the model is able to correct its mistakes when gets stuck at some point. Take figure 20b for example, the model made the first wrong move at the very beginning and therefore had a hard time making its way to the destination. However, it somehow crawled its way back, put itself on the right track and made it to the end. This is because the sampling process is bidirectional, meaning the paths between (A, B) and (A, B) are both included for training. There are cases where the model takes an entirely different path, as can be seen from figure 20d. There are cases where the model takes fairly good approximation of optimal paths, shown in figure 20a and figure 20c. More examples can be seen from appendix C.

5.2.4 Path evaluation

By visually comparing the paths traversed by the model with the optimal paths, many qualitative insights can be obtained. However, as discussed in section 3.3, it is also constructive to qualitatively measure the performance of the neural network using the proposed metrics.

It is interesting to see how the exploitation and exploration strategy (section 2.5.3) can be combined with shortest path and greedy strategy and how the graph traversal is affected. Instead of always choosing the next step inferred by the neural network or the edge with the smallest weight, the algorithm is given a probability p to randomly select from other available choices. If the number of available choices (i.e., available edges to pick) is denoted as N, then the model's choice or the edge with smallest weight is picked with probability $1-p$; other $N-1$ choices have probability $\frac{p}{N-1}$ to be picked each. When $p = 0.0$, the graph traversal follows pure shortest path or greedy strategy. The experiment is conducted on a new 100-node graph and 400 pairs of nodes are sampled. Different exploration rates are experimented with. For greedy strategy, 100 repeated test runs on those 400 node pairs are included and 50 test runs for shortest path strategy. Due to the page space limit, the resultd of only 25 test runs are plotted with boxplots.

The arrival rate (section 3.3) evaluation results are shown in table 6, figure 21a, and figure 22a. As argued previously, pure greedy strategy itself is not a good way to traverse a graph. A pure greedy strategy yields fairly low arrival rate in the test graph (0.01). However, with higher exploration rate (i.e., higher randomness, p), the arrival rate increases. This is expected since the use of exploration strategy helps the algorithm to escape dead loops. We can also see that when the randomness increases from 0 to 0.25, there is a significant increase of arrival rate (from 0.01 to 0.1733). However, the margin of increase from 0 to 0.25 is much smaller (only from 0.1733 to 0.2413). When the exploration rate increases from 0.50 to 0.75, there is no significant difference of arrival rate. This is interpretable in the sense that increasing randomness, at some point, will again confuse the model. It can be expected that there a sweet spot that leverages both exploitation and exploration, resulting in the maximum arrival rate for greedy strategy.

However, this is not the case for shortest path strategy. More bias towards exploration (i.e., high exploration rate) only yields lower arrival rate. This might be because when the model has a high accuracy on predicting the next step, introducing too much randomness only compromises its decisions. If we think with a real world analogy, when a person is good at solving multiple choice questions, it make sense to use skills and experience instead of flipping a coin. Based on this analogy, it is not surprising that the model achieves highest arrival rate when following its direct prediction (i.e., $p = 0.0$).

Figure 22b and figure 21b give the comparison of edge efficiency with shortest path and greedy strategy under different exploration rate. There is no significant difference of edge efficiency among different exploration rate, as can be seen from figure 22b. However, the edge efficiency for pure greedy strategy is 1. Further scrutiny shows that in this case, the paths found all have length 1, meaning the path consists of only two nodes and the edge weight happens to be the smallest. For shortest path strategy, there is a subtle rising trend of edge efficiency with the decreasing of exploration rate. When pure shortest path strategy is applied (i.e., $p = 0$), there is a boost of edge efficiency, which is expected from previous discussion.

Figure 22c and figure 21c show the comparison of time efficiency with shortest path and greedy strategy under different exploration rate. Similarly, there is no significant difference of time efficiency among different exploration rate, as can be seen from figure 22c. The time efficiency for pure greedy strategy is 1 for the same reason discussed previously. For shortest path strategy, there is also a subtle rising trend of time efficiency with decreasing exploration rate. One interesting observation is that there are less outliers when exploration rate decreases to 0.25 or 0. This might due to the increase of correctness of decisions made by the model. Also similarly, there is a boost of time efficiency when $p = 0$.

	Arrival Rate	Edge Efficiency	Time Efficiency
Shortest Path	1.0	1.0	1.0
Model $p = 75\%$	0.222 ± 0.021[21a]	Figure 21b	Figure 21c
Model $p = 50\%$	0.599 ± 0.026[21a]	Figure 21b	Figure 21c
Model $p = 25\%$	0.882 ± 0.020[21a]	Figure 21b	Figure 21c
Model $p = 0\%$	0.988 ± 0.007[21a]	Figure 21b	Figure 21c
Greedy $p = 75\%$	0.260 ± 0.022[22a]	Figure 22b	Figure 22c
Greedy $p = 50\%$	0.241 ± 0.026[22a]	Figure 22b	Figure 22c
Greedy $p = 25\%$	0.173 ± 0.018[22a]	Figure 22b	Figure 22c
Greedy $p = 0\%$	0.010 ± 0.000[22a]	Figure 22b	Figure 22c

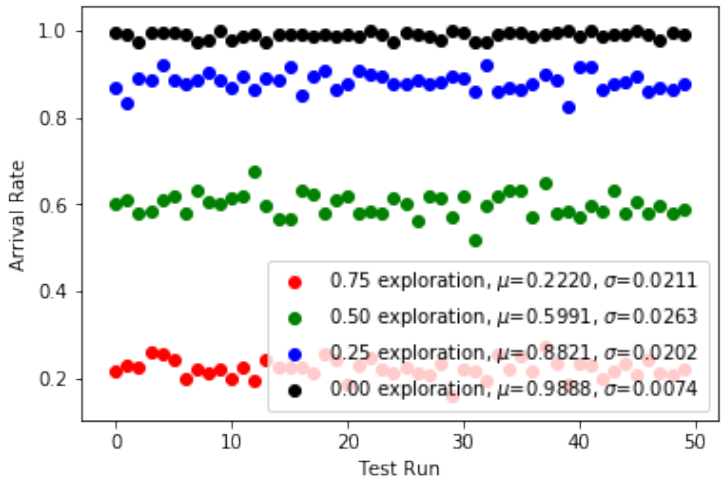
Table 6: A table shows the performance of different strategies.

5.2.5 Real time route planning

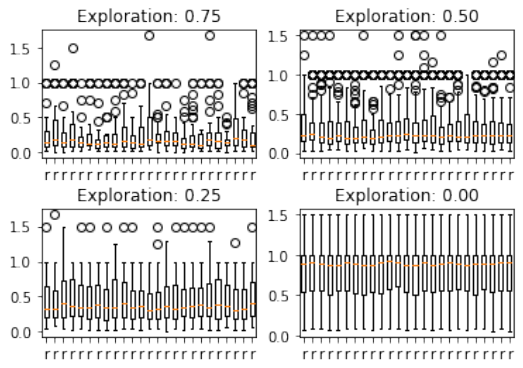
The dynamics of road networks has been discussed in section 1.2. Traditional shortest path algorithms fall short of capturing this dynamics, making real time routing inaccurate and computationally expensive. It is important that the effectiveness and path quality of the trained neural network are tested on real-time route planning. To simulate the dynamics of a road network, each time before the model makes a prediction on the next step, the weights of neighboring edges are modified according to an exponential distribution[18] with $\lambda = 1.0$. The motivation to choose this distribution is mainly that road conditions can vary drastically. However, it is much less likely that the road segment weights will double or triple. An exponential distribution describes this phenomenon well. An new graph is generated and 200 distinct node pairs are sampled. The routing experiment is repeated 50 times and arrival rate, time efficiency, and edge efficiency are computed accordingly. The results are shown in figure 23.

It is unexpected that the real time arrival rate is higher than that of static route planning, shown in figure 23a. In fact, three new graphs were generated and the same test routine was applied to them. The results were consistent with slightly different means and standard deviations. Figure 23a is selected as a representative. This might suggest that the trained model can indeed perform real time path finding tasks without sacrificing its effectiveness. Figure 23b shows the efficiency comparison between static and real time path finding. As can be seen, the time efficiency of static routing has

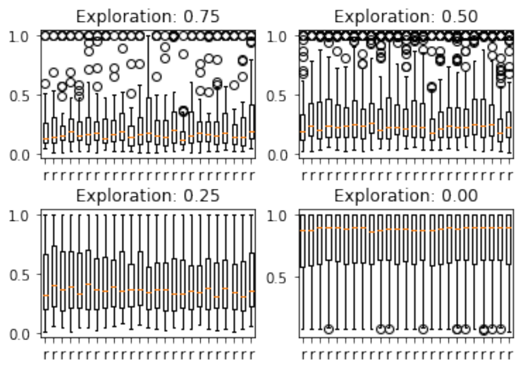
[18] `https://en.wikipedia.org/wiki/Exponential_distribution`



(a) A scatter plot shows the arrival rate of 4 exploration rate group tested on shortest path strategy.



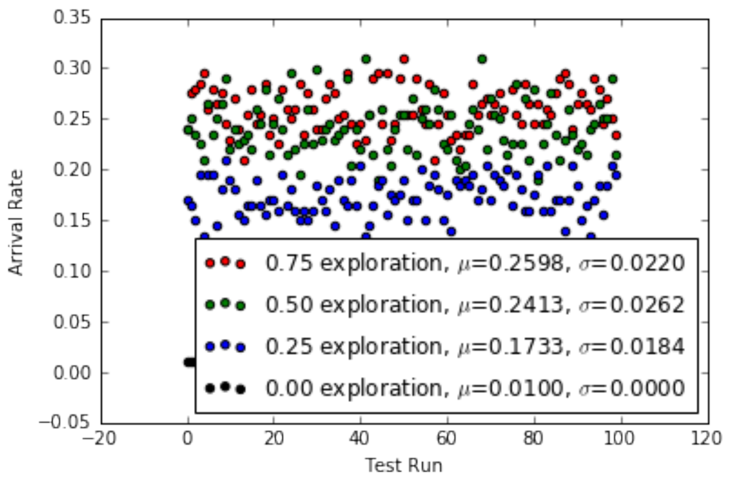
(b) Four boxplots show edge efficiency of shortest path strategy with different exploration rate.



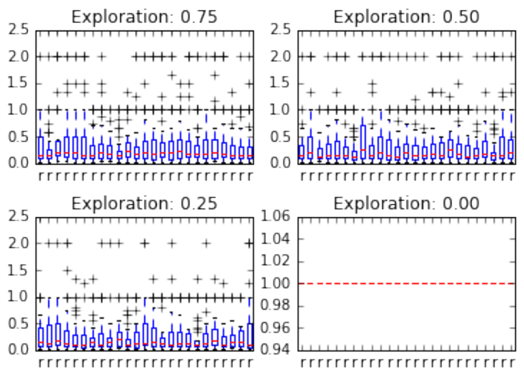
(c) Four boxplots show time efficiency of shortest path strategy with different exploration rate.

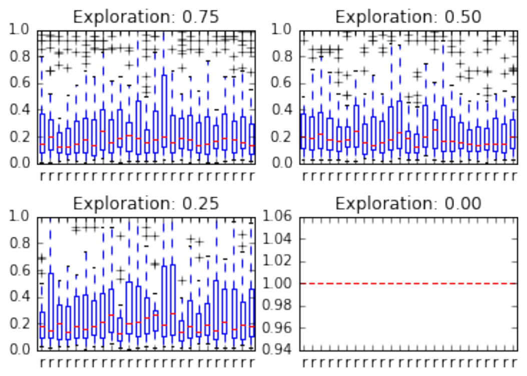
Figure 21: Three plots show the effectiveness and efficiency of shortest path strategy combined with exploration.

slightly higher median than that of real time routing. However, real-time time efficiency has wider range of lower and upper percentile and less outliers compared to static routing. This might suggest that real-time routing has less extreme performance issues and can output more smoothly. As can also be seen from figure 23b, the difference of edge efficiency between static and real-time routing is similar to that of time efficiency. The edge efficiency of static routing has much more outliers from both low and high end whereas



(a) A scatter plot shows the arrival rate of 4 exploration rate group tested on greedy strategy.



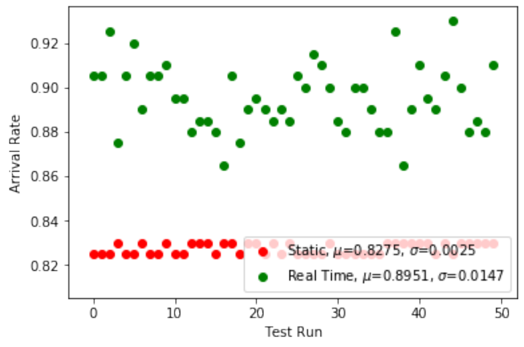


(b) Four boxplots show edge efficiency of greedy strategy with different exploration probability.

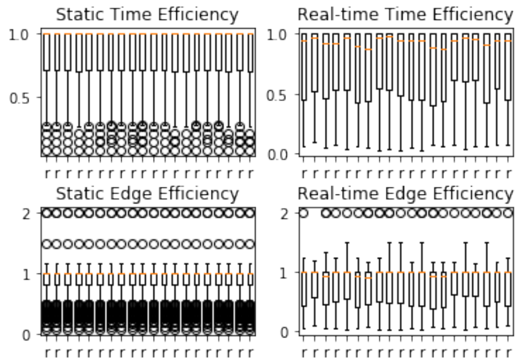
(c) Four boxplots show time efficiency of greedy strategy with different exploration probability.

Figure 22: Three plots show the effectiveness and efficiency of greedy strategy combined with exploration.

real-time routing has some outliers from the high value end. Similarly, the range of lower and higher percentiles in static routing is smaller than that of real-time routing. This might also suggest that real-time routing has more stable results.



(a) A plot shows the arrival rate comparison between static and real-time route planning.



(b) A plot shows the efficiency comparison between static and real-time route planning.

Figure 23: Two plots show the performance comparison between static and real-time route planning.

5.3 Road Network Learning

It is necessary that the insights and knowledge gained by training neural networks on random graphs can be well translated to road network. The target road network is shown in figure 11. The graph abstracted from this road network is shown in figure 15.

The path generated by the model is compared with actual shortest path, shown in figure 24. The black and green dots represent the origin and destination respectively. As can be seen, the model has applied similar strategies learned from the training on random graphs, seen from figure 20b. Even though the model made a wrong turn at some point and it struggle a bit by following the wrong direction, the model still managed to crawl its way back, put itself on the right track, and made its way to the destination. More examples can be seen from appendix D

In this section, only qualitative analysis is provided. However, the abstracted graph is in effect a simple graph and it is not necessarily more complicated or difficult for the a model to learn and traverse. As such, the performance of the neural network is not quantitatively measured on the road network. However, it can be expected that the performance of the neural network is at least on par with its performance on random graphs.

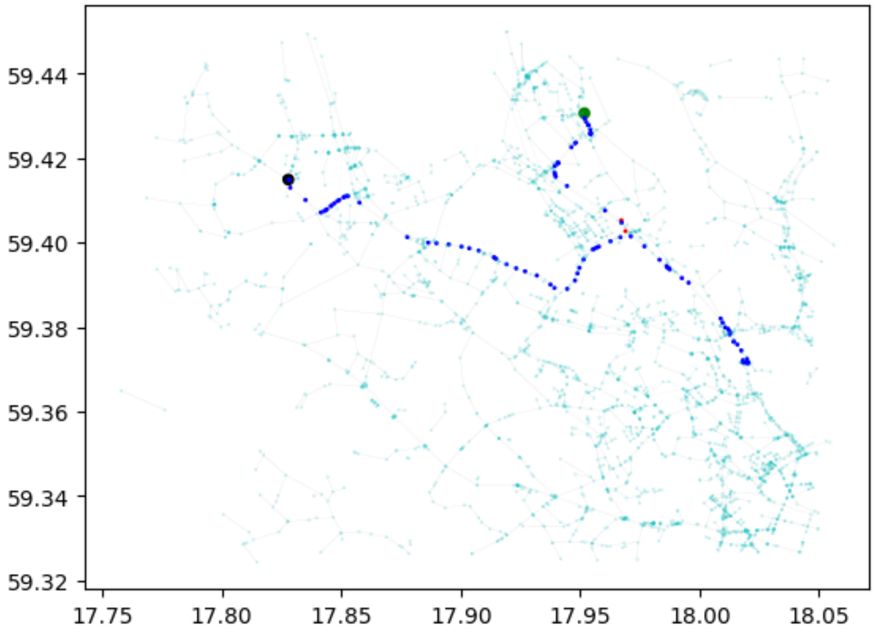


Figure 24: A plot compares the path generated by the model (blue) and by shortest path algorithm (red).

6 Conclusion

In this thesis work, we investigated if neural networks can be trained to perform real-time pathfinding tasks. The results give the answer to our research question,

Are neural networks applicable to real-time route planning tasks in a road network?

The effectiveness (i.e., arrival rate, section 3.3.1) of the trained model is around 90%. Ideally, we would expect the model to have an *arrival rate* of 100% so it always does its job. However, for a non-mission critical scenario such as pathfinding in a road network, a model with 90% of effectiveness can still be useful. Moreover, there is still room for the model to improve in terms of classification accuracy. Therefore, it is likely the effectiveness of the model can be boosted up to 95% or even 98%, which makes the model more applicable and useful. As for the path quality, the *time efficiency* (section 3.3.2) has a median of 0.88 and a relatively large variance. This indicates that the paths generated by the model are not stable. Much effort needs to be invested not only to improve the path time efficiency but also to guarantee the stability of the result. Meanwhile, The trained model shows potential for real-time route planning given that the model has better performance when dealing with varying weights of a road network. Overall, we can have a positive answer to the research question. This work shows that a neural network trained to make locally optimal choices can hardly give a globally optimal solution. We also show that our method, only making locally optimal choices, can adapt to dynamic graphs with little performance overhead.

6.1 Discussion

In this work, we started with the hypothesis that neural networks can be used for real-time route planning in road networks. A step-by-step approach is exploited during this work. We first designed an algorithm to train a neural network to be greedy, meaning it always makes the locally optimal choice. Then, the experience and knowledge were applied to train the model to be ”smart” about its decisions. We used various heuristics to guide the learning of the model, as discussed in section 4.3. The newly gained experience and knowledge were further translated to real road network preprocessing and training of the neural network.

The results from section 5.1 and 5.2 show that the model is not able to learn greedy and shortest path strategies well. As mentioned previously, Xu et al. [25] used three traffic parameter. Our approach includes more and can be easily extended to as many parameters as possible. However, our model has a high chance to yield suboptimal paths. There is a need to boost the model performance. Compared to Xu et al. [25] and Alex et al. [24], our approach uses a much simpler model architecture. The trained model is able to generalize well to large unseen graphs. The complexity of tasks performed in this study is much larger than the ones in [25, 24].

As discussed previously, shortest path algorithms have difficulty handling dynamic graphs. For a trained neural network, it makes locally optimal choices based on all local information and some global knowledge at that particular moment. This not only enables the model to keep moving forward but also makes the model less sensitive to the changes of a graph. The model has been trained to improvise in front of difficult situations. The experimental results in section 5.2.5 show that the model can adapt to the change of a road network without loss of effectiveness and path quality. Our method trains the model to make locally optimal choices; however, we realize that this hardly leads to a globally optimal solution. A better strategy should be adopted in order to have better results.

Meanwhile, the trained model might fail in other datasets. Since we customized the graph generation routine, graph properties such as weights, edge centralities are all generated according to some distribution. Other datasets might have different distributions hence the possible failure of the model. If we compare our method with Xu et al. [25] and Zhang et al. [28], we indeed adopted a simpler approach, however, their methods are guaranteed to be effective whereas our method has a chance to fail. The validity is enhanced by the fact that we have collected a large number of data points, and collectively they show that the model can be applicable to pathfinding in road networks, which again emphasizes the reliability of the results.

6.2 Future Work

As discussed previously, we generated data to test our hypothesis instead of using real traffic behavior. It would be constructive to train the model on real data. It will be interesting to see how accurately a full-blown route planning software package can perform. Meaningful and useful heuristics are explored but not limited by the aforementioned ones. It might be constructive to discovery more heuristics to further improve the pathfinding of the neural network. Meanwhile, different combinations of heuristics and how they affect the learning of the model are not investigated due to time constraint. It will be interesting to conduct feature selection and see how the model reacts.

To limit the scope of this study, the traversal behavior is only trained with simple multi-layer neural networks. There are many other techniques that show potential for learning strategies. Reinforcement Learning (RL) seems suitable for training a neural network to traverse a graph. It will be interesting to see how well a RL model can learn the greedy and shortest path strategy and compare its performance with the trained model in this work. There are also many good machine learning algorithms that can be trained on the data and compared with the model. The neural network is trained to make locally optimal choices. This means the model cares only about all local information and limited global information. Attention mechanism can be added to the neural network architecture to further guide its traversal.

We proposed and applied our own evaluation metrics. Are they really good metrics of the model? What other metrics are applicable? These questions require further investigation and more creativity. In section 5.2.5, the change of weights in a graph follows an exponential distribution with $\lambda = 1.0$. Is the value of λ applicable to the real world? Furthermore, does the exponential distribution factually reflect the dynamics of road networks? It takes more investigation to answer these two questions. As also discussed, the size of a road network is a limiting factor of our algorithm. As long as the road network reaches a certain size, it is infeasible to train the model on a full scale. Random sampling technique is applied in this work to make the algorithm tractable. This fact limits the use cases of our algorithm. In future work, it is important to make the algorithm more scalable for real use. In future work, the generalizability of the model trained on big graphs should be extensively tested.

References

[1] Tsapakis I, Cheng T, Bolbol A. Impact of weather conditions on macroscopic urban travel times. Journal of Transport Geography. 2013;28:204–211.

[2] Guzolek J, Koch E. Real-time route planning in road networks. In: Vehicle Navigation and Information Systems Conference, 1989. Conference Record. IEEE; 1989. p. 165–169.

[3] LeCun Y, Bengio Y, Hinton G. Deep learning. nature. 2015;521(7553):436.

[4] Hearst MA, Dumais ST, Osuna E, Platt J, Scholkopf B. Support vector machines. IEEE Intelligent Systems and their applications. 1998;13(4):18–28.

[5] Dumont M, Marée R, Wehenkel L, Geurts P. Fast multi-class image annotation with random subwindows and multiple output randomized trees. In: Proc. International Conference on Computer Vision Theory and Applications (VISAPP). vol. 2; 2009. p. 196–203.

[6] Breiman L. Random forests. Machine learning. 2001;45(1):5–32.

[7] Svetnik V, Liaw A, Tong C, Culberson JC, Sheridan RP, Feuston BP. Random forest: a classification and regression tool for compound classification and QSAR modeling. Journal of chemical information and computer sciences. 2003;43(6):1947–1958.

[8] Skiena S. Dijkstra's algorithm. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley. 1990;p. 225–227.

[9] Gubichev A, Bedathur S, Seufert S, Weikum G. Fast and accurate estimation of shortest paths in large graphs. In: Proceedings of the 19th ACM international conference on Information and knowledge management. ACM; 2010. p. 499–508.

[10] Katz GJ, Kider Jr JT. All-pairs shortest-paths for large graphs on the GPU. In: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware. Eurographics Association; 2008. p. 47–55.

[11] Ding B, Yu JX, Qin L. Finding time-dependent shortest paths over large graphs. In: Proceedings of the 11th international conference on Extending database technology: Advances in database technology. ACM; 2008. p. 205–216.

[12] Schultes D. Route Planning in Road Networks. In: Ausgezeichnete Informatikdissertationen; 2008. p. 271–280.

[13] Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics. 1968;4(2):100–107.

[14] Russell SJ, Norvig P. Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,; 2016.

[15] Garg D, et al. Dynamizing Dijkstra: A solution to dynamic shortest path problem through retroactive priority queue. Journal of King Saud University-Computer and Information Sciences. 2018;.

[16] Demetrescu C, Italiano GF. A new approach to dynamic all pairs shortest paths. Journal of the ACM (JACM). 2004;51(6):968–992.

[17] King V. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Foundations of Computer Science, 1999. 40th Annual Symposium on. IEEE; 1999. p. 81–89.

[18] Nannicini G, Liberti L. Shortest paths on dynamic graphs. International Transactions in Operational Research. 2008;15(5):551–563.

[19] Chan EP, Yang Y. Shortest path tree computation in dynamic graphs. IEEE Transactions on Computers. 2008;(4):541–557.

[20] Tamatjita EN, Mahastama AW. Shortest Path with Dynamic Weight Implementation using Dijkstra's Algorithm. ComTech: Computer, Mathematics and Engineering Applications. 2016;7(3):161–171.

[21] Black PE. Manhattan distance. Dictionary of Algorithms and Data Structures. 2006;18:2012.

[22] Xing EP, Jordan MI, Russell SJ, Ng AY. Distance metric learning with application to clustering with side-information. In: Advances in neural information processing systems; 2003. p. 521–528.

[23] Danielsson PE. Euclidean distance mapping. Computer Graphics and image processing. 1980;14(3):227–248.

[24] Graves A, Wayne G, Reynolds M, Harley T, Danihelka I, Grabska-Barwińska A, et al. Hybrid computing using a neural network with dynamic external memory. Nature. 2016;538(7626):471.

[25] Yongsheng X, Jianling W. Optimal path solution of urban traffic road. In: Natural Computation (ICNC), 2011 Seventh International Conference on. vol. 2. IEEE; 2011. p. 799–802.

[26] Brown J, Kovacs K, Vas P. A method of including the effects of main flux path saturation in the generalized equations of AC machines. IEEE transactions on power apparatus and systems. 1983;(1):96–103.

[27] Johnson JL, Ranganath H, Kuntimad G, Caulfield H. Pulse-coupled neural networks. In: Neural networks and pattern recognition. Elsevier; 1998. p. 1–56.

[28] Zhang Y, Wu L, Wei G, Wang S. A novel algorithm for all pairs shortest path problem based on matrix multiplication and pulse coupled neural network. Digital Signal Processing. 2011;21(4):517–521.

[29] Håkansson A. Portal of research methods and methodologies for research projects and degree projects. In: Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp); 2013. p. 1.

[30] Newman I, Benz CR. Qualitative-quantitative research methodology: Exploring the interactive continuum. SIU Press; 1998.

[31] Tarjan R. Depth-first search and linear graph algorithms. SIAM journal on computing. 1972;1(2):146–160.

[32] Bundy A, Wallen L. Breadth-first search. In: Catalogue of Artificial Intelligence Tools. Springer; 1984. p. 13–13.

[33] Pearl J. Heuristics: intelligent search strategies for computer problem solving. 1984;.

[34] Korf RE. Artificial intelligence search algorithms. Chapman & Hall/CRC; 2010.

[35] Lanning DR, Harrell GK, Wang J. Dijkstra's algorithm and Google maps. In: Proceedings of the 2014 ACM Southeast Regional Conference. ACM; 2014. p. 30.

[36] Floyd RW. Algorithm 97: shortest path. Communications of the ACM. 1962;5(6):345.

[37] Hougardy S. The Floyd–Warshall algorithm on graphs with negative cycles. Information Processing Letters. 2010;110(8-9):279–281.

[38] Cheng C, Riley R, Kumar SP, Garcia-Luna-Aceves JJ. A loop-free extended Bellman-Ford routing protocol without bouncing effect. In: ACM SIGCOMM Computer Communication Review. vol. 19. ACM; 1989. p. 224–236.

[39] Korf RE, Reid M. Complexity analysis of admissible heuristic search. In: AAAI/IAAI; 1998. p. 305–310.

[40] Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review. 1958;65(6):386.

[41] Hurford JR. Exclusive or inclusive disjunction. Foundations of language. 1974;11(3):409–411.

[42] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016. http://www.deeplearningbook.org.

[43] Nair V, Hinton GE. Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10); 2010. p. 807–814.

[44] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2016. p. 770–778.

[45] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:14091556. 2014;.

[46] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. nature. 1986;323(6088):533.

[47] LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, et al. Backpropagation applied to handwritten zip code recognition. Neural computation. 1989;1(4):541–551.

[48] Hecht-Nielsen R. Theory of the backpropagation neural network. In: Neural networks for perception. Elsevier; 1992. p. 65–93.

[49] Werbos PJ. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE. 1990;78(10):1550–1560.

[50] Riedmiller M, Braun H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: Neural Networks, 1993., IEEE International Conference on. IEEE; 1993. p. 586–591.

[51] Domingos P. A few useful things to know about machine learning. Communications of the ACM. 2012;55(10):78–87.

[52] Wang J, Perez L. The effectiveness of data augmentation in image classification using deep learning. Technical report; 2017.

[53] Tzanetakis G, Cook P. Musical genre classification of audio signals. IEEE Transactions on speech and audio processing. 2002;10(5):293–302.

[54] Takahashi N, Gygli M, Van Gool L. Aenet: Learning deep audio features for video analysis. IEEE Transactions on Multimedia. 2017;.

[55] Zhu J, Chen N, Perkins H, Zhang B. Gibbs max-margin topic models with data augmentation. Journal of Machine Learning Research. 2014;15(1):1073–1110.

[56] McFee B, Humphrey EJ, Bello JP. A Software Framework for Musical Data Augmentation. In: ISMIR. Citeseer; 2015. p. 248–254.

[57] Wong SC, Gatt A, Stamatescu V, McDonnell MD. Understanding data augmentation for classification: when to warp? In: Digital Image Computing: Techniques and Applications (DICTA), 2016 International Conference on. IEEE; 2016. p. 1–6.

[58] Medsker L, Jain L. Recurrent neural networks. Design and Applications. 2001;5.

[59] March JG. Exploration and exploitation in organizational learning. Organization science. 1991;2(1):71–87.

[60] Gelly S, Wang Y. Exploration exploitation in go: UCT for Monte-Carlo go. In: NIPS: Neural Information Processing Systems Conference Online trading of Exploration and Exploitation Workshop; 2006. .

[61] Chen J, Xin B, Peng Z, Dou L, Zhang J. Optimal contraction theorem for exploration–exploitation tradeoff in search and optimization. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans. 2009;39(3):680–691.

[62] Mehlhorn K, Newell BR, Todd PM, Lee MD, Morgan K, Braithwaite VA, et al. Unpacking the exploration–exploitation tradeoff: A synthesis of human and animal literatures. Decision. 2015;2(3):191.

[63] Alba E, Dorronsoro B. The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. IEEE transactions on evolutionary computation. 2005;9(2):126–142.

[64] Audibert JY, Munos R, Szepesvári C. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. Theoretical Computer Science. 2009;410(19):1876–1902.

[65] Davis N. The selfish gene. Macat Library; 2017.

[66] Tiakas E, Papadopoulos AN, Nanopoulos A, Manolopoulos Y, Stojanovic D, Djordjevic-Kajan S. Trajectory similarity search in spatial networks. In: Database Engineering and Applications Symposium, 2006. IDEAS'06. 10th International. IEEE; 2006. p. 185–192.

[67] Wang X, Tieu K, Grimson E. Learning semantic scene models by trajectory analysis. In: European conference on computer vision. Springer; 2006. p. 110–123.

[68] Zhou Y, Yan S, Huang TS. Detecting anomaly in videos from trajectory similarity analysis. In: Multimedia and Expo, 2007 IEEE International Conference on. IEEE; 2007. p. 1087–1090.

[69] Dubuisson MP, Jain AK. A modified Hausdorff distance for object matching. In: Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on. vol. 1. IEEE; 1994. p. 566–568.

[70] Bunke H. On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters. 1997;18(8):689–694.

[71] Watts DJ, Strogatz SH. Collective dynamics of 'small-world'networks. nature. 1998;393(6684):440.

[72] Brandes U. A faster algorithm for betweenness centrality. Journal of mathematical sociology. 2001;25(2):163–177.

[73] Lerner J, Wagner D, Zweig K. Algorithmics of large and complex networks: design, analysis, and simulation. vol. 5515. Springer; 2009.

[74] Goodwin LC. Weather impacts on arterial traffic flow. Mitretek systems inc. 2002;.

A Dijkstra's and A* search Algorithm

Algorithm 1: Dijkstra's and A* Search General Framework

Input: **G** denotes the graph, **Src**, **Dst**

Output: Shortest path between **Src** and **Dst** or **FAILURE**

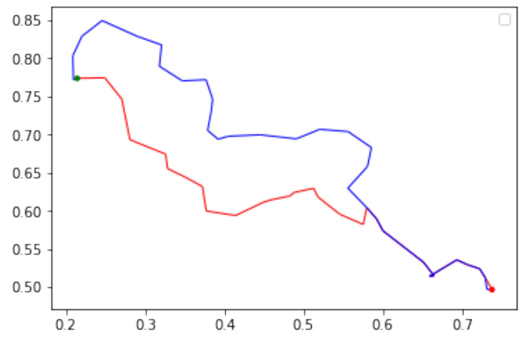
```
1  Let Set_closed denote the nodes that already been evaluated;
2  Let Set_open denote the discovered nodes that have not been evaluated;
3  Let path_map record the predecessor of each node with regard to shortest
     path;
4  Let g record g(n) with all node initialized as INFINITY;
5  Let g(Src) be equal to 0;
6  Let f record the total cost and the value for each node initialized as
     INFINITY;
7  Let f(Src) be equal to h(Src);
8  while Set_open is not empty do
9      Retrieve the node from Set_open that has the lowest f score, denoted as
         cur_node;
10     if cur_node is Dst then
11         return the shortest path reconstructed from path_map.
12     end
13     Remove cur_node from Set_open;
14     Add cur_node to Set_closed;
15     for Each neighbor of cur_node not in Set_open do
16         Add neighbor to Set_open;
17         tentative_g = g(cur_node) + edge_weight(cur_node, neighbor);
18         if tentative_g $\geq$ g(neighbor) then
19             # No better path;
20             continue;
21         end
22         # This path is better path_map <- (neighbor, cur_node);
23         g(neighbor) <- tentative_g;
24         f(neighbor) <- g(neighbor) + h(neighbor);
25     end
26 end
27 return FAILURE;
```

B Software Used

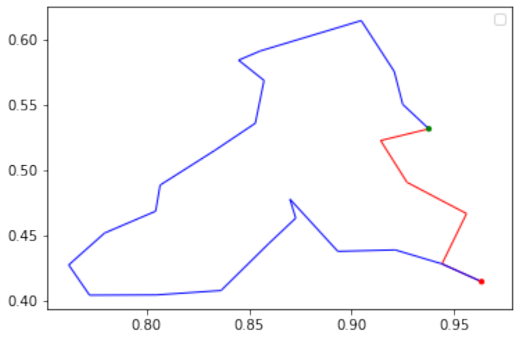
Software	Use Case	Version
NetworkX	Graph processing in python	2.1
Matplotlib	Plot functions in python	1.3.1
NumPy	Process arrays	1.8.0rc1
MXNet	Deep learning framework to train models	1.0.0
Scikit-learn	Machine learning algorithms	0.19.1
Docker	Deploy code on both VM and local machines	17.09.0
Jupyter Notebook	Code running environment	4.4.0
OpenStreetMap.jl	OSM api for Julia	0.8.2
LightGraphs.jl	Graph processing	0.12.0
GraphPlot.jl	Visualize graphs	0.2.0
PyPlot.jl	Provide plot functions	2.5.0
PyCall.jl	Connect Python and Julia	1.16.1
StatsBase.jl	Statistics package in Julia	0.22.0
DataFrames.jl	Process dataframe file	0.11.6
CSV.jl	Process CSV file	0.2.4
SUMO	Investigate real simulation software	0.32.0

Table 7: A table shows the software used during the experiments

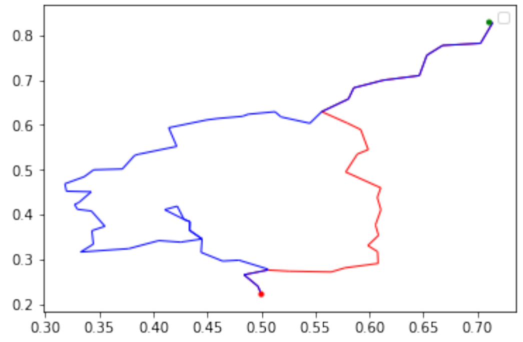
C More Examples of Path Comparison



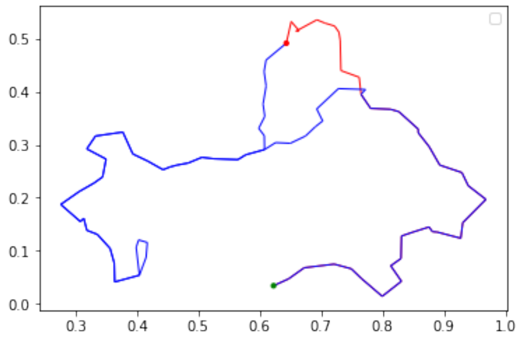
(a) 11



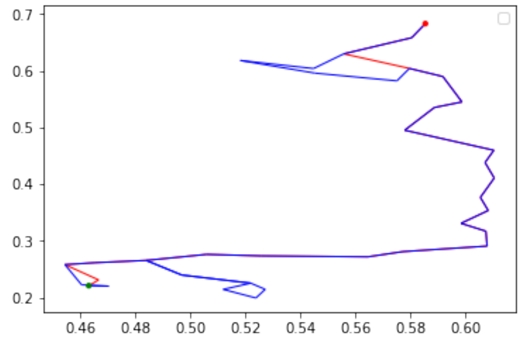
(b) 12



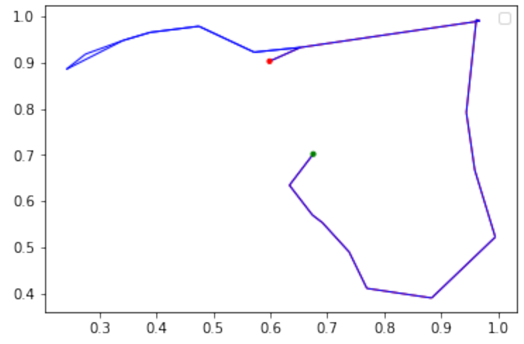
(c) 21



(d) 31



(e) 31



(f) 32

Figure 25: Plots show more examples of path comparison.

D More Examples on Road Network Paths Comparison

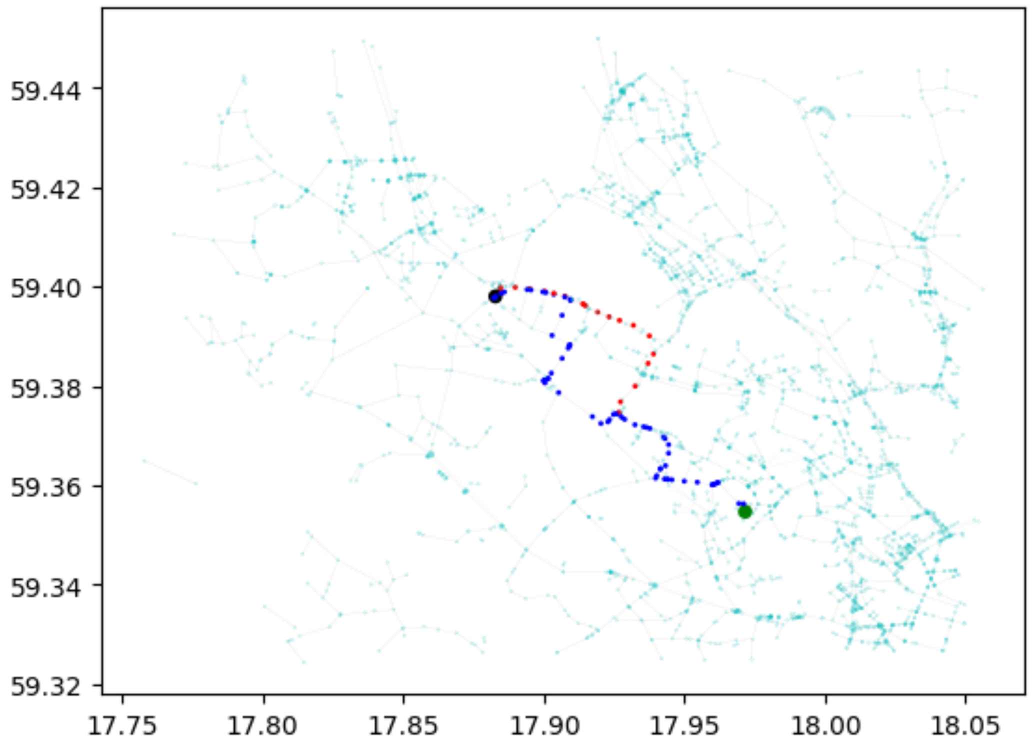


Figure 26: A plot compares the path generated by the model (blue) and by shortest path algorithm (red).

TRITA EECS-EX-2018:556