KTH Information and
Communication Technology

Deep Learning Models for Route Planning in Road Networks

TIANYU ZHOU

Master Thesis at KTH Information and Communication Technology Department
Supervisor: Sarunas Girdzijauskas
Examiner: Henrik Boström
Industrial Supervisor: Robert Luciani (LakeTide AB, Sweden)

TRITA-ICT-EX-2018:nn

Abstract

The purpose of this study is to expand the domain of route planning in road networks with neural agents (i.e., Neural Network models). Traditional algorithms can efficiently find the optimal paths in road networks under certain constraints. This study is to investigate how neural agents can be used to effectively and efficiently find paths in a road network.

A staggered approach is applied in progressing this investigation. The first step is to generate random graphs to facilitate the initial study. The next step is to determine, as a proof of concept, if a neural agent could 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 route planning in actual road networks.

Experimental results indicate that shortest path algorithms can be approximated with neural agents effectively and efficiently. Neural agent has the potential to be used for real time route planning in road networks.

Keywords: Route Planning, Road Networks, Neural Networks.

Sammanfattning

Syftet med denna studie är att utöka domänen för ruteplanering i vägnät med neurala medel (dvs Neural Network-modeller). Traditionella algoritmer kan hitta de optimala vägarna i vägnät under vissa begränsningar. Denna studie är att undersöka hur neurala medel kan användas för att effektivt och effektivt hitta vägar i ett vägnät.

Ett förskjutet tillvägagångssätt tillämpas under pågående undersökning. Det första steget är att generera slumpmässiga grafer för att underlätta den inledande studien. Nästa steg är att, som ett bevis på konceptet, bestämma om ett neuralt medel kunde lära sig att korsa enkla grafer med flera strategier, eftersom vägnätverk är i praktiken komplexa grafer. Slutligen uppskalar vi genom att inkludera faktorer som kan påverka ruteplaneringen i faktiska vägnät.

Experimentella resultat indikerar det Kortaste banalgoritmer kan approximeras med neurala medel effektivt och effektivt. Neurala medel har potential att användas för vägplanering i vägnät.

Nyckelord: Ruttplanering, Vägnät, Neural Networks.

Acknowledgements

I am grateful to those who helped me during the entire process 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 want to express my special gratitude to my supervisor, Professor Sarunas Girdzijauskas and examiner, Professor Henrik Boström who have been patiently guiding my thesis writing process.

Tianyu Zhou

July 11, 2018

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Route planning, i.e., path finding, in road networks aims to find the optimal paths between a pair of specified origin and destination [3]. 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 less 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 need to be considered. Secondly, the expansion of modern cities and road networks makes the route planning more and more computationally expensive. Thirdly, road network, as a dynamic system, makes it difficult for traditional route planning, which treats the road network as static. It is challenging, if not possible to make real time route planning. 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 computing models. Provided with considerable amount of training data, Neural Networks are able to learn to perform well on various tasks [4]. Neural Network can be potentially trained to traverse a road network, taking into account factor such as road length, speed, and current traffic. It is debatable that there are many other Machine Learning algorithms that are suitable such as Support Vector Machine (SVM) [5], Decision Tree (DT) [6], and Random Forest (RF) [7]. SVM, RF, and DT have been successful in the industry for many years [8].

The motivation for choosing Neural Network is twofold. Firstly, it is acknowledged that Neural Networks achieves state-of-the-art in many tasks with much less feature engineering. This is due to its ability to extract high-

level features from data automatically. Secondly, it is computationally difficult for traditional Machine Learning algorithms such as SVM to be trained on large dataset [9, 10]. However, Batch Gradient Descent [11] makes it possible to train Neural Networks on large dataset in reasonable time hence better performance.

1.2 Problem

Previously, the challenges encountered by route planning are listed. There have been much effort seeking to overcome these challenges. To conquer the first challenge, different criteria are treated differently by algorithms with various priorities. For most users who rely on online map services such as Google Map and Bing Map, the minimum traveling time is of first priority. The most commonly used algorithm for path finding is Dijkstra's algorithm that 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. It also computes the shortest paths from single source to all other nodes, which is a large waste. Its most popular variation, A* search algorithm, uses heuristics to speed up the search. The time complexity of A* search is $O(E)$. The considerable speedup comes with the price that the paths can be sub-optimal. The use of heuristics in A* search algorithm requires certain constraints. It is even more challenging if not possible, to combine many factors such as road segment properties, traffic conditions and formulate the heuristic that satisfies those constraints. Furthermore, A* search algorithm falls short of capturing the dynamic essence of road networks. Stated differently, once a path is computed and followed by a vehicle, the changes of road segments might make the path invalid. One way to overcome this is to compute the path between the current location to the destination every time a crossroad is reached, which in turn makes the route planning computationally intensive. An algorithm has large practical use if it can yield high-quality paths without sacrificing computing time and can also make real time local optimal decisions. There is potential for the path finding to be more dynamic, taking into account road network properties, road conditions, and other factors that typically affect road network states.

Neural Network approach for route planning in road networks has been applied in many studies. Jindal et al. [12] tried to estimate travel time and

distance with Neural Network model. Mikluščák et al. [13] uses Neural Networks to predict the routes and destinations of a vehicle based on its historical routes. These two studies are useful in many other aspects of road network applications but they did not directly conduct route planning in road networks. JF Gilmore et al. [14] used Neural Network to improve route planning for robots. The application of this study is constrained to 2-D robotic motion planning. The motion of the robot was also limited to a grid. The study from Alex et al. [1] shows that Neural Networks can indeed be trained to perform path finding tasks. However, the work done by Alex et al. is limited to traverse simple graph like London Underground map [1] and the complexity of traversal tasks is small. Real road networks are far more complex and contain richer information. An important work that sets ground for this thesis work is from Y. Xu et al. [15]. This works used road segment length, path saturation [16], and vehicle speed to generate weights for each road. Neural Networks were further applied to select the optimal path according to the weights. However, this work still considered a limited set of variables. There is room for more factors to be considered. Moreover, the experiment of this study is on an undirected graph with 16 nodes, which is apparently oversimplified. Its generalizability to large road networks remains untested.

Traditional route planning algorithms such as A* search have been proven to be effective and efficient. There are also attempts to further push the performance limit of traditional algorithms using Neural Networks. The knowledge gap remains to investigate how Neural Network can be trained with more factors so that it can effectively and efficiently find paths in a road network. Therefore, the research question is formulated as the following:

To what extent can a Neural Network be trained to perform route planning tasks in a road network effectively and efficiently?

By effectiveness and efficiency, the neural agent will be evaluated with the metrics discussed in section 3.2. Further literature study shows that this research question exhibits multiple aspects:

1. It is not self-evident how a neural network agent should traverse arbitrary graphs [2], let alone road networks. As such, initial investigation needs to build the ground for path finding in arbitrary graphs with neural agents.

[1] `http://content.tfl.gov.uk/standard-tube-map.pdf`

[2] `https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)`

2. What factors (e.g., traffic density, road properties, road popularity etc.) strongly affect the path finding of neural agents?
3. Do trained neural agents generalize well to new graphs?
4. Do the found principles and gained insights in simple graphs translate well to road networks?

1.3 Purpose & Goals

In this study, we want to use neural agents to perform path finding tasks so that it is real-time, flexible, fast, generalizable, and can guarantee path quality. To bridge the knowledge gap, the following smaller goals are set:

1. Neural agents are trained to traverse a simple graph with greedy strategy (section 4.1).
2. More sophisticated shortest path strategy is trained on neural agents (section 4.2).
3. The experience gained from the first step is translated to real road networks to see how the neural agents perform (section 4.3).

1.4 Methodology

This thesis work follows the usual steps of scientific research, forming a hypothesis, formulating research questions and setting up goals, implementing software to fulfill the goals, designing evaluation processes, conducting deeper discussions, and reaching a conclusion. A wide literature study is conducted to gain knowledge about the domain of traffic simulation regarding how it is conducted and what the challenges are.

A deductive study works from general cases to more specific ones [17, 18]. Deductive approach is used to compose the hypothesis from existing studies on route planning, shortest path algorithms, and Neural Networks. We generate data or use existing data to validate our hypothesis. In this study, both quantitative and qualitative [17, 19] methods are applied to evaluate the neural network agents.

1.5 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 an computational agent 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 [3], 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.

1.6 Delimitations

As with all studies, this work is constrained by time. This thesis work is limited not to build a full-blown route planning software package. Instead, a simple prototype is implemented and the learnability of path finding behavior is the main focus. In this study, a subset of Stockholm road network from OpenStreetMap [4] is studied. The generalizibility of neural network agents to other road networks will not be investigated. This work does not aim to compare which machine learning algorithms are more suitable for learning on road networks. As also discussed, the power of Neural Networks is a strong motivation to make it the main focus of this study. Therefore, the comparison of different machine learning algorithms will not be provided. Nevertheless, the performance of different algorithms can be investigated in future work. The principle of Occam's Razor [5] is strictly followed so that the scope of this study is well defined. The Neural Network topology is kept as simple as possible.

[3] https://www.un.org/sustainabledevelopment/sustainable-development-goals/
[4] https://www.openstreetmap.org
[5] https://en.wikipedia.org/wiki/Occam%27s_razor

1.7 Outline

The rest of the thesis is organized as the following:

- *Theoretical Background* will have further discussion of shortest path algorithms, and Neural Network theories. Related work is also extensively introduced.
- *Method* elaborates the design choices, the approaches used, and steps taken to answer the research questions.
- *Results* present the experimental settings and procedures. Corresponding results are presented and further elaborated.
- *Discussions* will contain a further discussion about the results. Findings and insights will be shared.

2 Theories & Related Work

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 [20] and breadth-first search [21]. 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 [22].

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 [23, 24]. "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 [6] and Bing Map [7] rely on shortest path algorithms [25]. In this study, we use the output of shortest path algorithms as the ground truth when training neural network agents and the benchmark for evaluation. Heuristics guided shortest path algorithms will be extensively discussed and applied to guide the training of neural network agents.

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 [26]. Floyd-Warshall, Bellman-Ford, Dijkstra's, and A* search are the most popular algorithms for finding the

[6] https://www.google.com/maps
[7] https://www.bing.com/maps

shortest paths. Floyd-Warshall algorithm [27] 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 [28] 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 [29] 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 [30] 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 [31]. 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 [32] 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 output scalar value y such as the following:

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

where w is vector of real values, seen from figure 1. This computation is essentially an affine transformation [8].

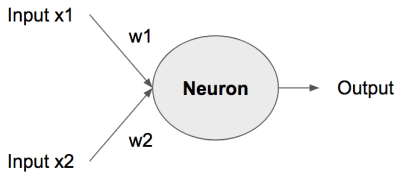


Figure 1: An illustration of Perceptron.

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 [33], which is defined in the following table:

[8] 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. However, there are certain traits of a non-linear 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 [34].

The most widely used non-linear functions are hyperbolic tangent function (i.e., tanh), rectified linear unit (i.e., ReLU [35] 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 [36] that has a depth of up to 152 layers. This is eight times deeper than VGG nets [37]. 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:

$$MSE(y, \hat{y}) = (y - \hat{y})^2$$

Cross Entropy Loss

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

$$CE(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$$

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 [34].

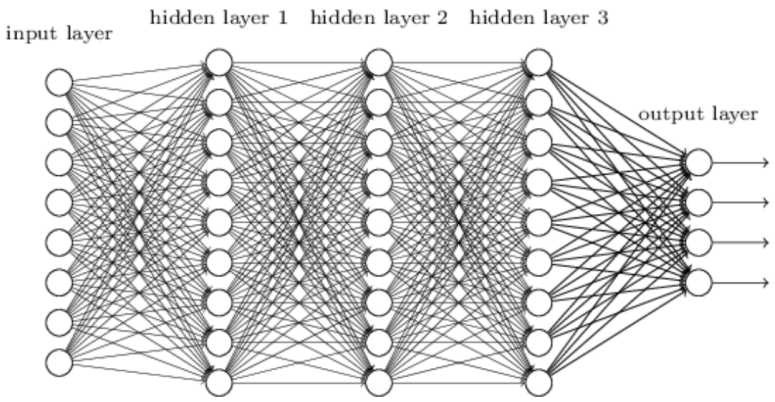


Figure 2: The Architecture of a fully-connected feedforward network.

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 [38] 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 [39, 40, 41, 42]. 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.

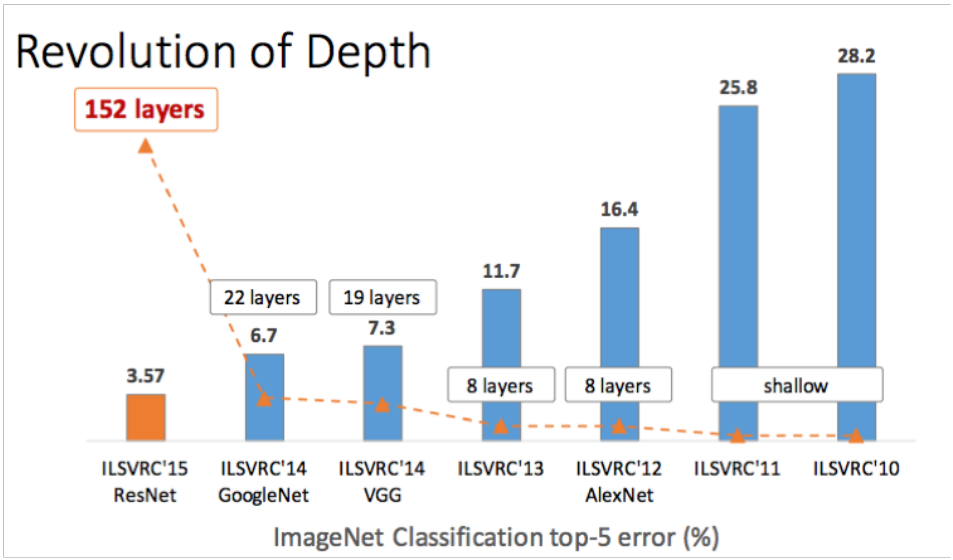


Figure 3: A plot shows the revolution of depth.

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 [9]. 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 neural network agents

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

can 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 [43]. When training size is small, there is no obvious advantage of using neural networks.

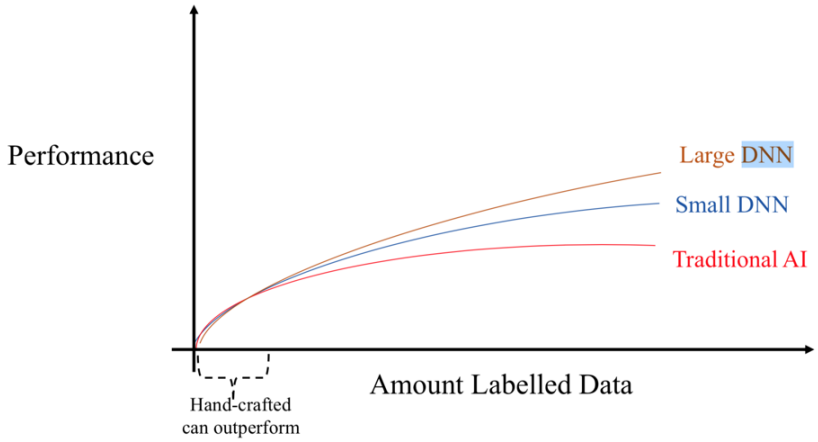


Figure 4: A comparison of traditional machine learning algorithms and Deep models.

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, distort-

ing, adding noise, cropping are commonly used techniques [44]. Similarly, noise, distortion can also be added to video and audio [45, 46] 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 [47, 48].

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 [49]. 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.

2.5 Related Work

2.5.1 Differentiable neural computers

With the remarkable capability of approximating arbitrary functions, neural networks still have difficulty representing complex data structures and storing data over the long term due to the lack of external memory [1]. To bridge this gap, Alex et al. [1] proposed a novel neural network architecture with a dynamic external memory, i.e., Differentiable Neural Computer (abbr., DNC). Modern computer architecture separates computation and memory. CPU or GPU, as the processing unit, is responsible for the computation whereas memory is used to stored the intermediate results. DNC is given access to a read-write external memory, which is a $N \times M$ matrix that can be trained along with the neural networks with gradient descent and more importantly, be persisted for later use.

For the model to solve graph problems, Alex et al. trained it on randomly generated graphs shown in figure 5 with curriculum learning [50, 51]. Then, the model was evaluated on real graph dataset, London Underground map [10], as shown in figure 5. The trained model executed traversal and shortest path

[10] `http://content.tfl.gov.uk/standard-tube-map.pdf`

tasks, yielding an average accuracy of 98.8% on traversal tasks and an average accuracy of 55.3% on shortest path tasks.

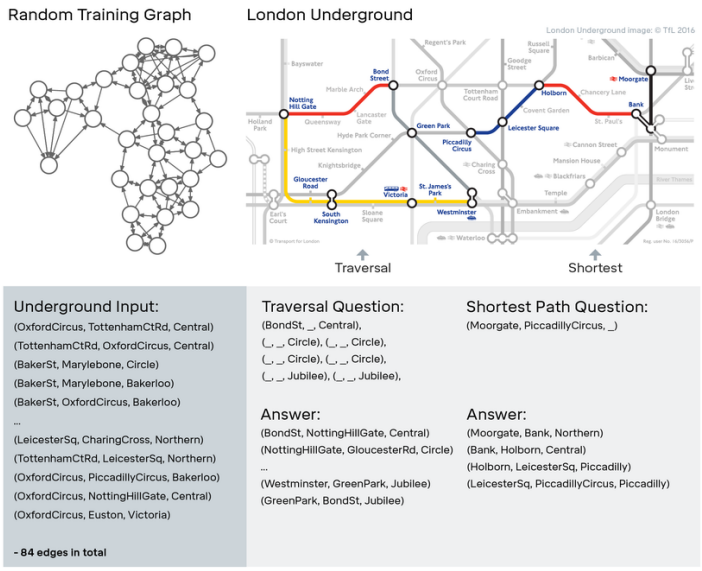


Figure 5: The picture from [1] shows the training and inference of DNC.

The result is encouraging since it is a direct proof that deep learning model is indeed able to traverse a simple graph. However, this does not mean that there is no need for further investigation in this study. There is room for further discussion.

1. The new architecture is indeed powerful yet very complex. As a first attempt in this study, it might be overkill to use such an architecture. Moreover, the encouraging results are hardly interpretable. As shown in the paper, the learning and inference of the model does not require any graph knowledge. This is against the purpose of this work, which is to investigate the learning properties of neural network agents.
2. The graph traversal and shortest path tasks performed on London Underground map is oversimplified. The traversal tasks are constrained by four-step traversal and shortest path tasks are limited under seven-step paths in the map. Such controlled behavior rarely happens in road network traversal and path finding. This motivates us to continue this thesis work.

2.5.2 Optimal path with traffic data fusion

In the work done by Y. Xu et al. [15], only three traffic parameters are considered, path saturation, vehicle speed, and road length. We believe that a road network is more than three parameters. Meantime, Y.Xu et al. used D-S evidence theory [52] to fuse these parameters, resulting in the weight of each road segments. Then the optimal paths are computed by a Pulse Coupled Neural Network (PCNN) [53]. Still, there is need for further discussion about this work:

1. Parameters are limited to only three. However, more can be included such as road popularity (indicated by road centrality), road geographic information such as distance and direction.
2. The parameters are fused by an algorithm that incorporates human knowledge. The output is then fed to a Neural Network. It was not properly motivated why data fusion was used not to mention that fusion process might cause information loss.
3. Moreover, the use of PCNN was not properly argued. Indeed, PCNN can be powerful in tasks such image processing and segmentation [54], motion detection [55] and so on. It does not necessarily mean that only PCNN can take into account the dynamics of road traffic condition.

To summarize, there is a need to investigate if a simple Neural Network can take more parameter into account and still perform reasonably well. The execution of this thesis project will be elaborated in the next chapter.

3 Methodology

As introduced previously, this study takes a staggered approach and is reasonably divided into three steps, as shown in figure 6.

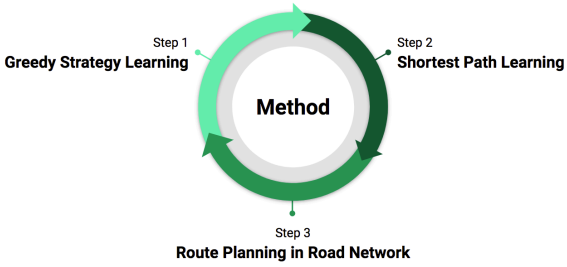


Figure 6: A circle shows three stages of this study.

The execution of each stage will be elaborated in the following sections. This chapter is organized as the following: The design choices made and difficulties encountered in the thesis work are discussed in section 3.1. These difficulties and choices might convey important insights and serve as useful guidance for future research. In section 3.2, we will talk about the customized metrics that evaluate the performance of neural agents. In section 3.3, the procedures implemented to generate data for neural agent training and evaluation are discussed. Finally, the learning of two strategies on neural agents and translation to the road network are elaborated in section 4 and section 4.3 respectively.

3.1 Design Choices

3.1.1 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. [1] and Y. Xu et al [15]. 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 [11][12]. As a proof of concept for our purpose, spending too 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 (abbr., RNN) [56] 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.

3.1.2 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

[11] `https://github.com/deepmind/dnc` [1]
[12] `https://github.com/pcnn/traffic-sign-recognition`

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.

3.1.3 Exploration & exploitation

The exploration and exploration strategy has been applied in various fields such as sociology, biology, business development, organization science [57], search and optimization algorithms [58, 59]. The classic definitions of exploration and exploitation given by James G. March [57] 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 [60, 61, 62, 59, 58, 57]. 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 [63], 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. The neural agents exploit the graph by deterministic inference. As developers, we need to help the agents to explore by randomizing the output with a certain probability (e.g., 10%) and see how that affects the paths.

3.2 Evaluation Metrics

This thesis work does not aim to perfectly approximate shortest path algorithms but to investigate how factors such as road traffic, road properties affect the paths generated by agents. Given that shortest path algorithms are used to generate the ground truth and benchmark, we need to quantitatively measure how the agents perform. Path similarity can be categorized as trajectory similarity measures. Measuring trajectory similarity has many important use cases in various fields [64, 65, 66]. 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 [67]. 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 7, which is due to the fact that Hausdorff distance does not take continuity of paths into consideration.

2. Fréchet distance; As shown in figure 8, all points a in curve A is mapped to the points in curve B through continuous function $\mu : A \rightarrow B$. Then, the distance between curve A, B is:

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

Figure 7: A picture shows an edge case where two very different paths have small Hausdorff distance

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

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

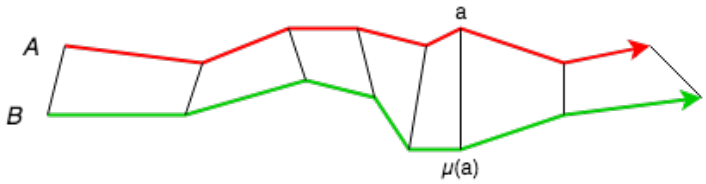


Figure 8: A picture visualizes the calculation of Fréchet distance

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 [68]. A simple example is shown in figure 9. 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.

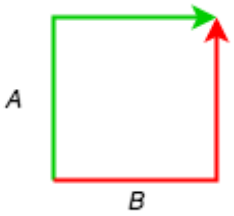


Figure 9: A picture is used to explain Edit distance with the grid hidden.

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 the case, a path generated by neural agent can still be very good but classified as a "bad" path.

3.2.1 Motivations

When it comes to comparing the neural network agent generated path with shortest path, two aspects are of great interest:

1. Effectiveness: can a agent find a path? What is the probability that an agent can find a path from A to B in a graph?
2. Efficiency: can a agent find a path efficiently? The path can have either less steps or smaller sum of edge weights.

3.2.2 Destination arrival rate

After the agent 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 agent to give a path between them. If the agent has a high chance to give a path to the queries, we can treat it as a workable agent. The arrival rate can be defined as the following:

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

3.2.3 Path weight sum

Drivers care about how fast they can reach their destinations. It is meaningful to know how efficiently neural agents can make it to the destination. 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 a agent as $path_{agent}$. The time efficiency of the agent is:

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

Smaller the sum of edge weights of the path, the more efficient the agent is.

3.2.4 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 an agent has to take to find the destination. We can define *edge efficiency* as the following:

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

3.3 Data Collection

3.3.1 Random graph generation

Real road networks are complex and hard to process. A commonly used open source map tool is OpenStreetMap [13]. 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 10. The graph can be generated as either directed or undirected.

[13] `http://openstreetmap.org`

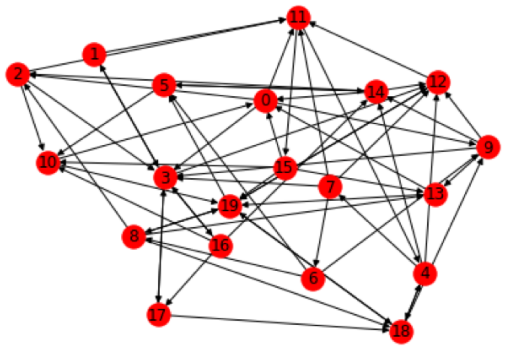


Figure 10: A random graph has 20 nodes.

Shortest path strategy

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 11. More complex cases such as roundabout might emerge. It is not much more complicated and can be adapted correctly.

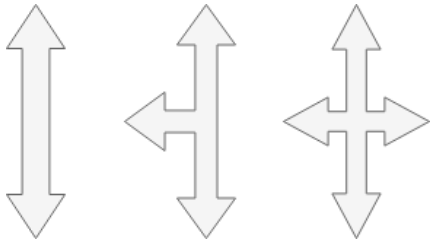
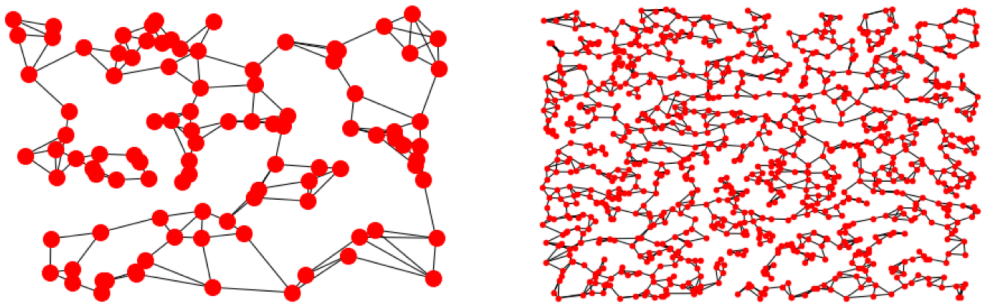


Figure 11: A example shows the out degrees of a crossroad in a road network.

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 12: Two examples show the output of random graph generation procedure

Two graphs can be seen from figure 12a and figure 12b. An observation is that the random graph with 1000 nodes presents certain small-world traits [69]. 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.

3.3.2 OpenStreetMap

OpenStreetMap (OSM) is a free editable world map that is collaborated by many people [14]. People can edit and download the map data from the website, as shown in figure 13. A screenshot of northern part of Stockholm is

Figure 13: A screenshot of northern Stockholm from OpenStreetMap.

shown in figure 13 [15]. This area has the bounding box of $min\_longitude = 17.7755, max\_longitude = 18.0464, min\_latitude = 59.3274, max\_latitude = 59.4421$.

OSM provides APIs for many programming languages. In this work, we mainly use the Julia OSM API [16]. With the API, the road network can be conveniently extracted from the map data, which is essentially a XML file.

[14] `https://en.wikipedia.org/wiki/OpenStreetMap`
[15] `https://www.openstreetmap.org`
[16] `http://openstreetmapjl.readthedocs.io/en/stable/`

3.4 Experimental Settings

The hardware used for the experiments is shown in table 1. The hardware is host on Azure Virtual Machine [17]. The main programming languages used in this work is Python 2.7 [18] and Julia 0.6 [19]. 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 [20]

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

The model is evaluated with accuracy. The 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, the metrics discussed in section 3.2 are applied.

[17] https://azure.microsoft.com/en-us/services/virtual-machines/
[18] https://www.python.org/download/releases/2.7/
[19] https://julialang.org/downloads/
[20] https://github.com/zhoutianyu16tue/PathFinder

4 Graph Traversal Strategies

4.1 Greedy strategy

To imitate human behavior, 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 local 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 agent could learn the greedy strategy. All code is available from github [21].

4.1.1 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 14, $edge(1, 3)$ 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:

[21] `https://github.com/zhoutianyu16tue`

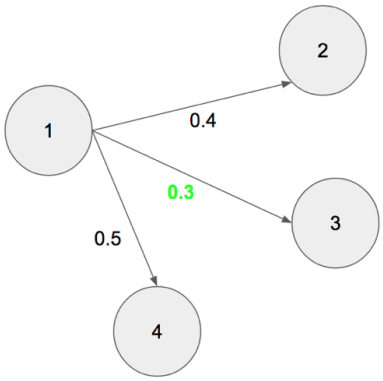


Figure 14: An example shows picking the edge with smallest weight for a node.

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

    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
```

4.1.2 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.

4.2 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.

4.2.1 Local knowledge

Edge weights are essential for a neural agent to learn shortest path strategy. Edge weights alone do not effectively guide the search of a neural agent. 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 [70]. 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

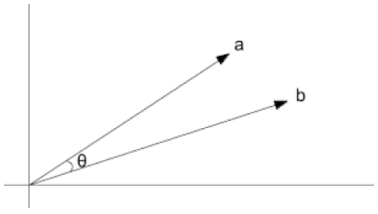
shortly, popular roads attract more vehicles and hence more traffic congestions, which in return affect human driving behavior.

4.2.2 Global knowledge

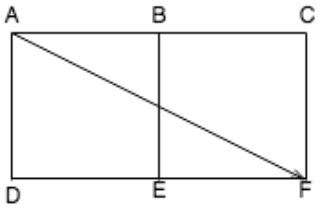
Two main global heuristics are provided to the neural agents, geographic distance (i.e., euclidean distance) and cosine distance. These two indicators reflect the two questions that a human drive would frequently ask:

- 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 15: 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 15a. The mathematical form is the following:

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

Figure 15b 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.

4.2.3 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 [71] 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 agents. It must be emphasized that the paths generated by the agents are not necessarily the optimal ones. The aim to is to train the agent to make local optimal choices so the time complexity is constrained by $\Theta(E)$ and guarantee the path quality.

4.2.4 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 16. 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 agents 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 16, $Edge(1, 3)$ 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.

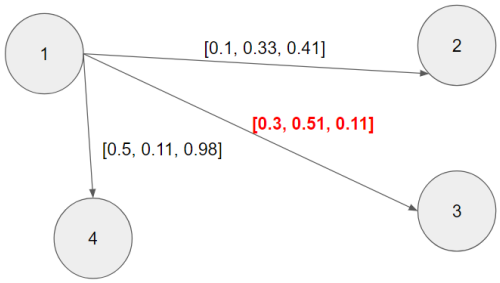


Figure 16: An example shows picking the edge according to A* search

4.2.5 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 14 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 16 can be arranged arbitrarily, as long as the label indicates which edge should be picked.

4.3 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 3.3.1.

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 13 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.

4.3.1 Road network processing

The road network that can be abstracted from figure 13 is shown in figure 17. It is worth noticing that the granularity of a road network that can be extracted from OpenStreetMap can be configured manually [22]. 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 17, 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 17. 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.

4.3.2 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 neural agent.

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

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

as global knowledge. Weather significantly affects traffic conditions [2]. 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 showing how weather increases travel time [2].

win [72], 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 3.3.1. 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 17: A plot illustrates the road network abstracted from figure 13.

5 Results

5.1 Greedy Strategy Learning

5.1.1 Training and inference

As discussed in section 4.1, each node can be represented as a 1-d vector, shown in figure 14. 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 4.2.5).

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 18.

As can be seen from figure 18, 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 19. Apparently, the model trained with 100-node graph has better

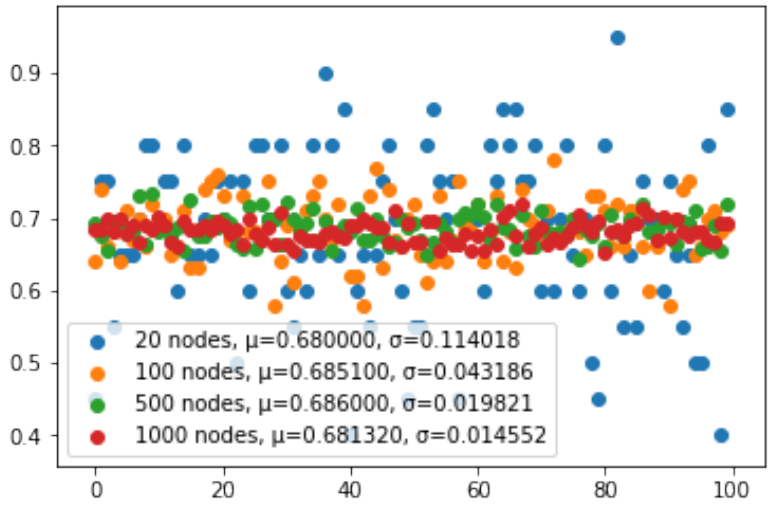


Figure 18: A plot illustrates the generalizability of a model trained on a graph of 20 nodes.

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

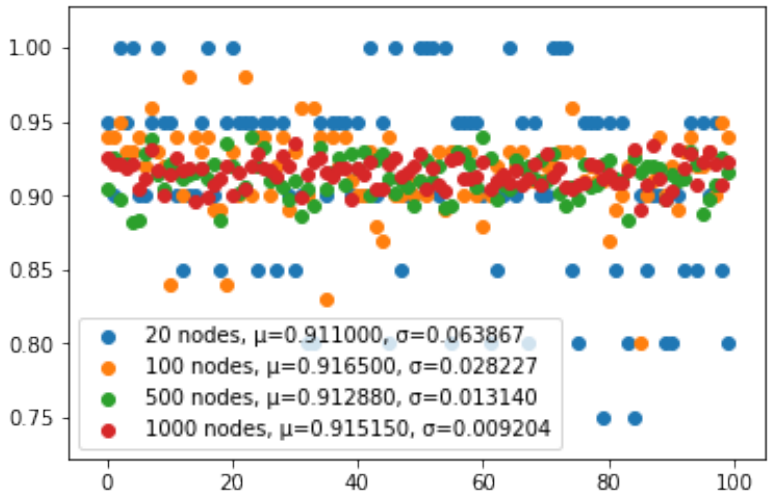


Figure 19: A plot illustrates the generalizability of a model trained on a graph of 100 nodes.

Data augmentation technique 4.2.5 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 20. 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.

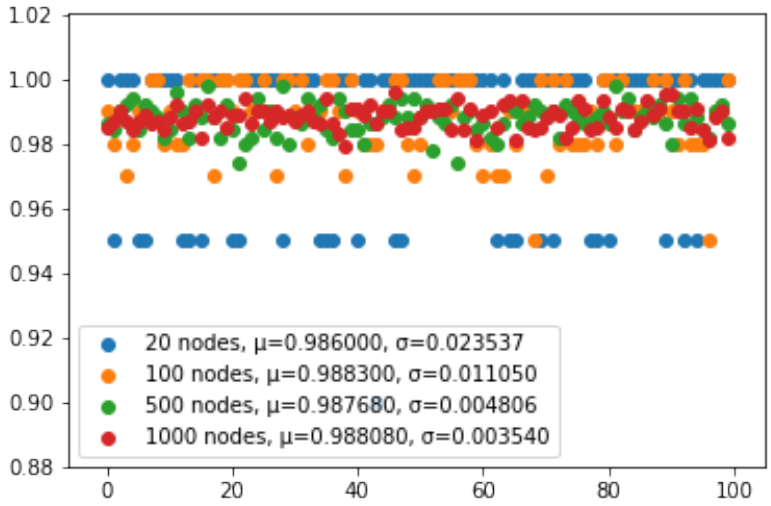


Figure 20: A plot illustrates the generalizability of a model trained on a graph of 100 nodes with data augmentation.

5.2 Shortest Path Learning

5.2.1 Neural agent 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 agent 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 agent is shown in table 4. The (x, y) coordinates of the source and destination are also provided as a global knowledge (section 4.2) 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 3.3.1, 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 agent 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 3.1.3, the neural agent should not only exploit the available information but also explore the graph with randomness, which is introduced by its mistakes.

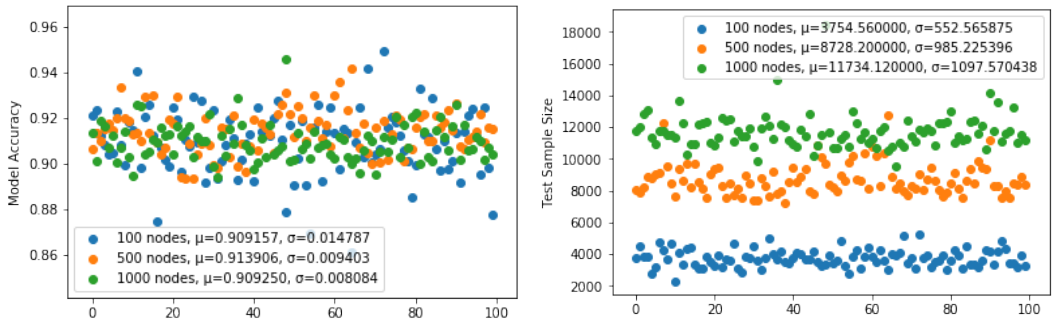
5.2.2 Model generalizability

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 21.

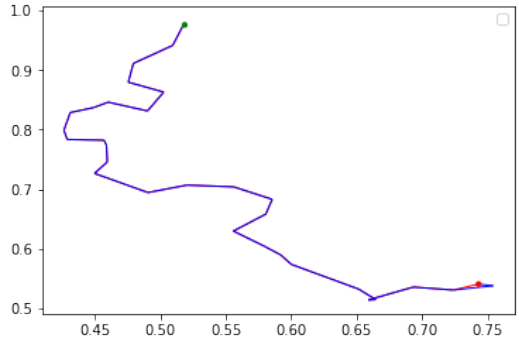


(a) A plot shows the model accuracy distribution on each size group.

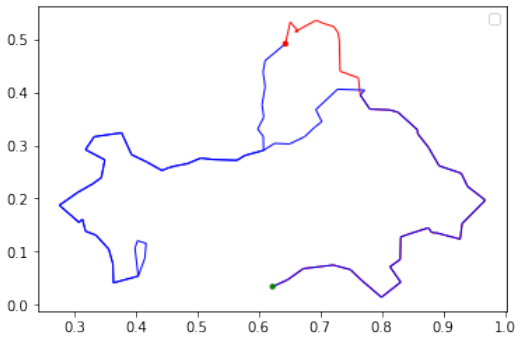
(b) A plot shows the sample size statistics over each run.

Figure 21: Two plots show the result of the shortest path strategy learning generalizability test.

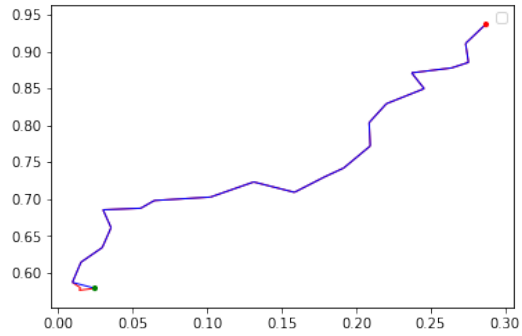
As can be seen from figure 21b, it is expected a bigger graph can generate more samples with the same number of node pairs. The average length of paths is linear to the size of a graph. As also shown in figure 21a, the model is able achieve the same level of accuracy with acceptable standard deviation 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.



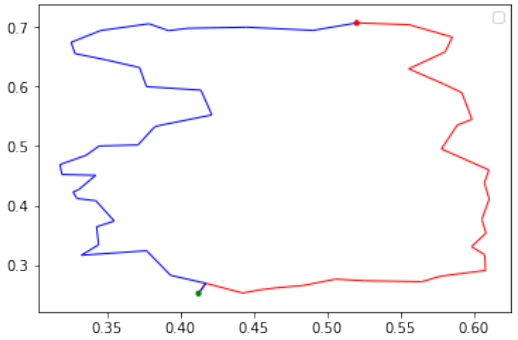
(a) A plot shows a good match between two paths.



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



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



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

Figure 22: Plots show the comparison between agent generated paths and shortest paths.

5.2.3 Path visualization

The trained model is applied on the 1000-node shown in figure 12b. The paths generated by the neural agent is compared with the corresponding optimal paths. The plots are shown in figure 22. Those four plots are selected carefully and therefore are representative.

Apparently, those paths are far from being optimal. One interesting observation is that the agent is able to correct its mistakes when gets stuck at some point. Take figure 22b for example, the agent 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 22d. There are cases where the model takes fairly good approximation of optimal paths, shown in figure 22a and figure 22c. More examples can be seen from appendix C.

5.2.4 Path evaluation

By visualizing the paths traversed by neural agents with shortest paths, many qualitative insights can be obtained. However, as discussed in section 3.2, it is also constructive to qualitatively measure the performance of neural agents using the proposed metrics.

It is interesting to see how the exploitation and exploration strategy (section 3.1.3) 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 agent 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 agent'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 3.2) evaluation results are shown in table 6, figure 23a, and figure 24a. 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 greedy strategy since the greedy agent is traversing the graph with less rules but only more randomness. 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 agent 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 agent achieves highest arrival rate when following its direct prediction (i.e., $p = 0.0$).

Figure 24b and figure 23b 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 24b. 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 24c and figure 23c 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 24c. 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 agent. 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
Agent $p = 75\%$	0.222 ± 0.021[23a]	Figure 23b	Figure 23c
Agent $p = 50\%$	0.599 ± 0.026[23a]	Figure 23b	Figure 23c
Agent $p = 25\%$	0.882 ± 0.020[23a]	Figure 23b	Figure 23c
Agent $p = 0\%$	0.988 ± 0.007[23a]	Figure 23b	Figure 23c
Greedy $p = 75\%$	0.260 ± 0.022[24a]	Figure 24b	Figure 24c
Greedy $p = 50\%$	0.241 ± 0.026[24a]	Figure 24b	Figure 24c
Greedy $p = 25\%$	0.173 ± 0.018[24a]	Figure 24b	Figure 24c
Greedy $p = 0\%$	0.010 ± 0.000[24a]	Figure 24b	Figure 24c

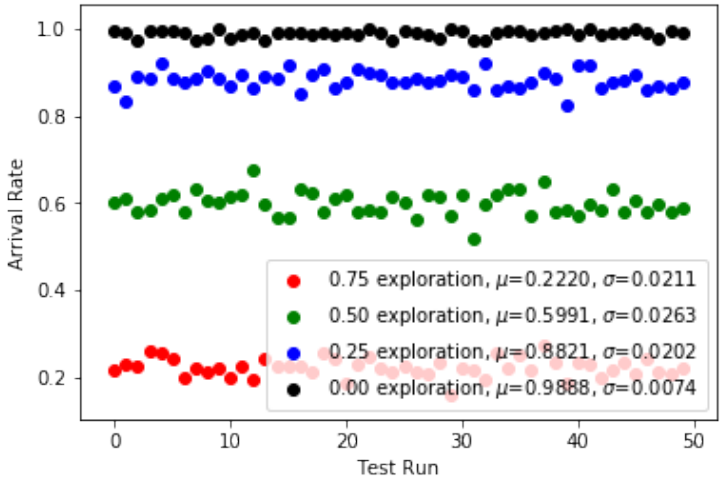
Table 6: A table shows the performance of different strategies.

5.2.5 Real time route planning

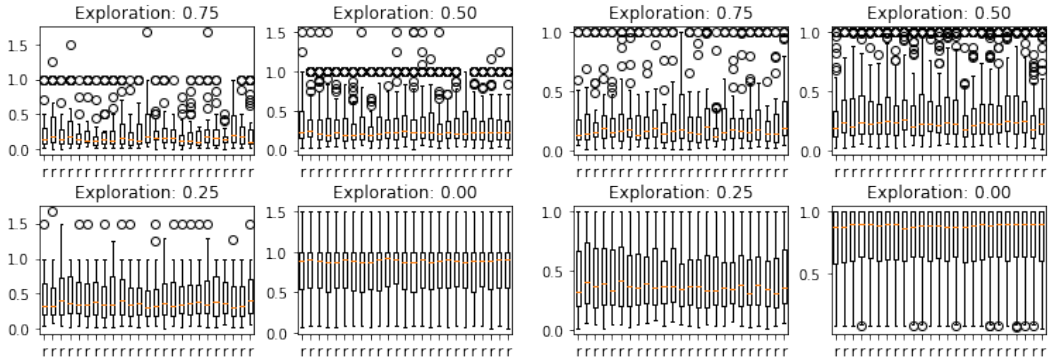
The dynamics of road networks has been discussed in section 1.2. Traditional shortest path algorithms fall short of capturing this dynamics, making real time routing inaccurate and computationally expensive. It is important that the effectiveness and efficiency of the trained neural agent are tested on real-time route planning. To simulate the dynamics of a road network, each time before the agent makes a prediction on the next step, the weights of neighboring edges are modified according to an exponential distribution [23] 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 25.

It is unexpected that the real time arrival rate is higher than that of static route planning, shown in figure 25a. 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 25a is selected as a representative. This might suggest that the trained agent can indeed perform real time path finding tasks without sacrificing its effectiveness. Figure 25b shows the efficiency comparison between static and real time path finding. As can be seen, the time efficiency of static routing has

[23] https://en.wikipedia.org/wiki/Exponential_distribution



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

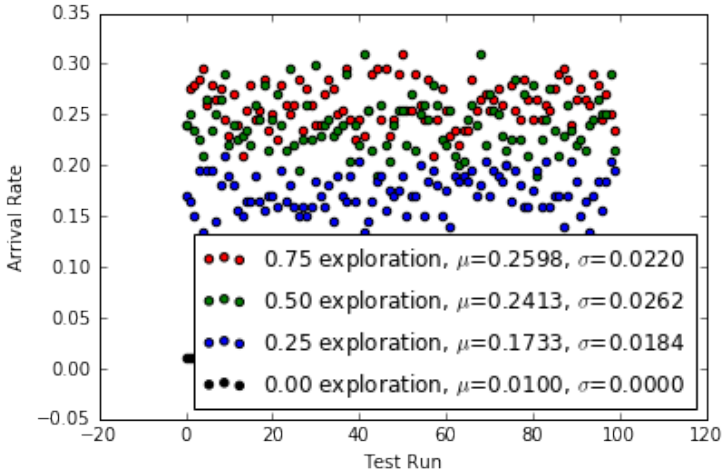


(b) Four boxplots show edge efficiency of shortest path strategy with different exploration rate.

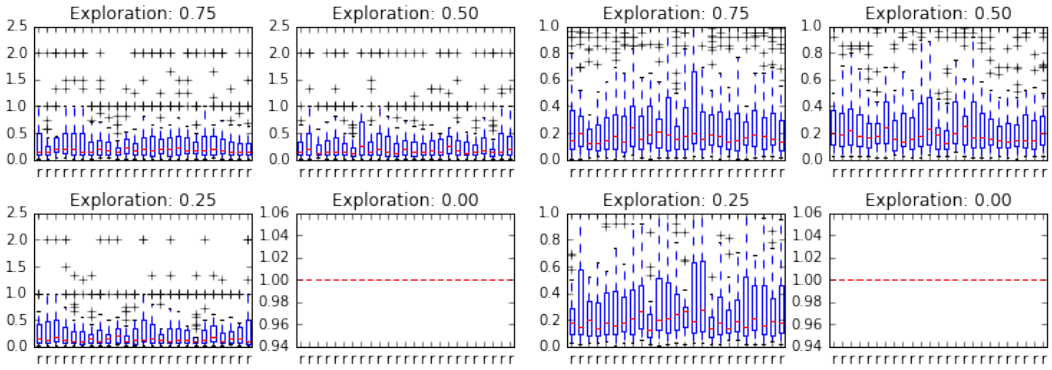
(c) Four boxplots show time efficiency of shortest path strategy with different exploration rate.

Figure 23: 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 25b, 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 arrive rate of 4 exploration rate group tested on greedy strategy.

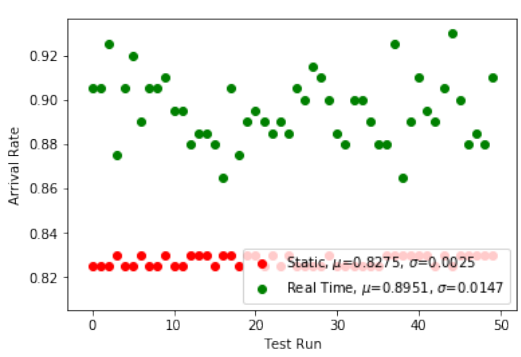


(b) Four boxplots show edge efficiency of greedy strategy with different exploration probability.

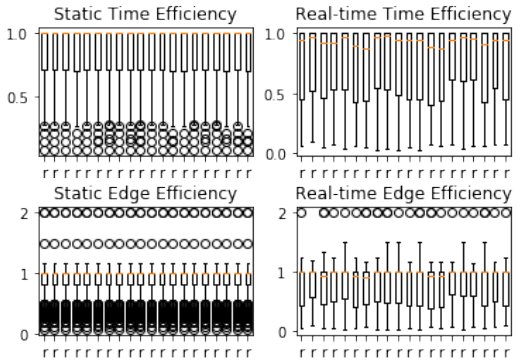
(c) Four boxplots show time efficiency of greedy strategy with different exploration probability.

Figure 24: 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 25: Two plots show the performance comparison between static and real-time route planning.

5.3 Road Network Learning

Previous evaluation has shown that neural agents can indeed traverse a graph effectively and efficiently. 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 13. The graph abstracted from this road network is shown in figure 17.

The path generated by the neural agent is compared with actual shortest path, shown in figure 26. The black and green dots represent the origin and destination respectively. As can be seen, the agent has applied similar strategies learned from the training on random graphs, seen from figure 22b. Even though the agent made a wrong turn at some point and it struggle a bit by following the wrong direction, the agent 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 neural agent is not quantitatively measured on the road network. However, it can be expected that the performance of the neural agent is at least on par with its performance on random graphs.

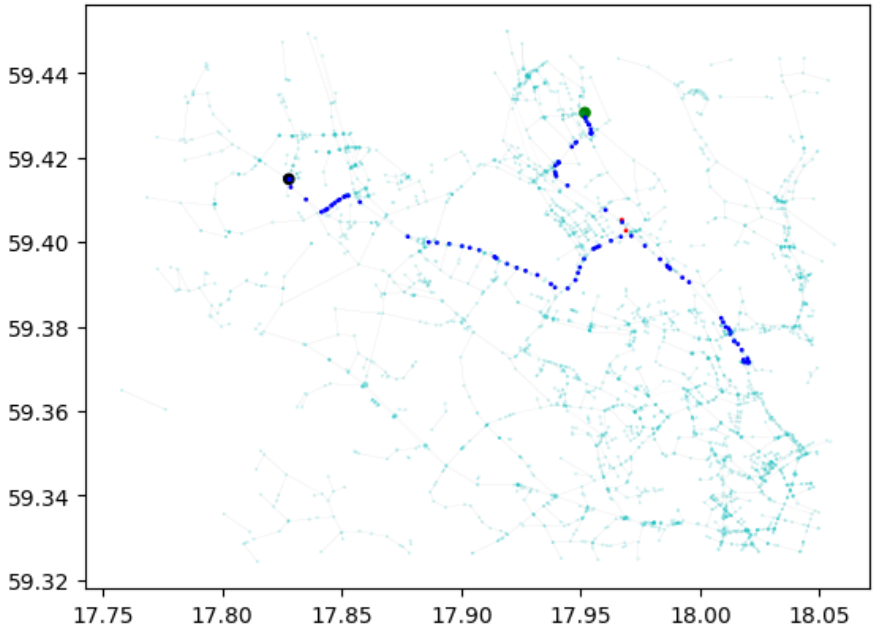


Figure 26: A plot compares the path generated by the agent (blue) and by shortest path algorithm (red).

6 Discussions

6.1 Findings

In this work, we started with the hypothesis that Neural Networks can be used for route planning in road networks. A step-by-step approach is exploited during this work. We first designed an algorithm to train a neural agent to be greedy, meaning it always makes the local optimal choice. Then, the developing experience and knowledge were applied to train the agent to be ”smart” about its decisions. We used various heuristics to guide the learning of the neural agent, as discussed in section 4.2. The newly gained experience and knowledge as a developer were further translated to real road network preprocessing and training of the neural agent.

The results from section 5.1 and 5.2 show that the agent is not only able to learn both greedy and shortest path strategies but also generate paths that are reasonable. The agents can also generalize well to unseen scenarios. This makes it feasible to train models efficiently and to use them in various use

cases. When the model does not take the actual optimal paths, it does not necessarily put a bad sign. Shortest path algorithms have no concept of time. This means they treat a graph as if it is static. To get the actual shortest paths when running these algorithms, the decisions are made once and are based on the global view of the graph. When the weights on some edges change, the algorithms need to restart in order to give the correct results. For trained neural agents, they make local optimal choices based on all local information and some global knowledge at that particular moment. This not only enables the agent to keep moving forward and but also make the agent less sensitive to the changes of a graph. When it comes to traffic, a ever-changing graph is not friendly to traditional shortest path algorithms whereas neural agents have been trained to improvise in front of difficult situations. The experimental results in section 5.2.5 show that the neural agent can adapt to the change of a road network without too much performance loss. In contrary, neural agent might be more effective (i.e., higher arrival rate) in real-time routing compared to state road network routing.

6.2 Challenges

During this thesis work, constant challenges have been encountered. They include how to start the work, how to obtain and use data, and most importantly, how to evaluate.

At the beginning of the project, the problem itself was not clearly defined. What does it mean by simulating traffic with a Neural Network model? What can possibly be the input and output of the model? What can be the network topology (discussion in section 3.1. To start something simple turned out to be the savior. Not only is the first strategy simple but also the model architecture could not be simpler. When the first trial worked, it further encouraged us to explore more complex strategies and to be creative about how to guide the training of an agent.

When it came to obtaining data, things became tricky. On the one hand, real traffic data contains sensitive personal geo-location information, which makes it hard to obtain and use. On the other hand, this data truly reflects real world route planning, which makes it the perfect source for training the agent. Following the principle of start with something simple and build upon that, we designed random graph generation procedures that can approximate real road networks and can be still kept simple, as discussed in sec-

tion 3.3.1. Approaching the end of the project, real data has not been used for training due to its sensitivity. However, it can be expected that the model can be even more real, flexible, and smarter once trained on real path finding data.

To complete the circle of this study, evaluation is necessary. Traditional Machine Learning evaluation metrics such as accuracy, f1-score, or mean square error do not seem to be applicable in our study. However, both qualitative and quantitative methods occurred to us with the process of the project. By combining the traits of route planning and past personal traveling experience, a few evaluation metrics (discussion in section 3.2) were proposed and applied.

To summarize, this project has been both challenging and fun. As discussed in section 3.1.3, the algorithms are design to both exploit and explore. This perfectly summarized the experience of working on this project. We not only leverage the present resource and knowledge but also try to be creative and innovative.

6.3 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 aforementioned ones. It might be constructive to discovery more heuristics to further improve the path finding of neural agents. 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 an agent react.

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 (abbr., RL) seems suitable for training an agent to traverse a graph. It will be interesting to see how well a RL agent can learn the greedy and shortest path strategy and compare its performance with the neural agent in this work. There are also many good machine learning algorithms that can be trained on the data and compared with the neural agents. Neural agents are trained to make

local optimal choices. This means the agent cares only about all local information and limited global information. Attention mechanism can be added to the Neural Network topology to further guide its search.

We proposed and applied our own evaluation metrics. Are they really good metrics of the neural agents? What other metrics are applicable? This 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 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 agent 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 agents trained on big graphs should be extensively tested.

6.4 Conclusion

At the end of this report, it is not to be made conclusive. Personal experiences and insights are discussed and shared with hope that it serve as a guideline for future studies. In this thesis work, we investigated how Neural Networks can be trained to perform path finding tasks. The results give the answer to our research question, i.e., Neural Networks can indeed be trained to effectively and efficiently find good-quality paths in a road network without much computational overhead. The trained neural agent can potentially be used for real-time route planning tasks.

References

[1] 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.

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

[3] 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.

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

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

[6] 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.

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

[8] 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.

[9] Menon AK. Large-scale support vector machines: algorithms and theory. Research Exam, University of California, San Diego. 2009;117.

[10] Lulli A, Oneto L, Anguita D. Crack random forest for arbitrary large datasets. In: Big Data (Big Data), 2017 IEEE International Conference on. IEEE; 2017. p. 706–715.

[11] Wilson DR, Martinez TR. The general inefficiency of batch training for gradient descent learning. Neural Networks. 2003;16(10):1429–1451.

[12] Jindal I, Chen X, Nokleby M, Ye J, et al. A Unified Neural Network Approach for Estimating Travel Time and Distance for a Taxi Trip. arXiv preprint arXiv:171004350. 2017;.

[13] Mikluščák T, Gregor M, Janota A. Using neural networks for route and destination prediction in intelligent transport systems. In: International Conference on Transport Systems Telematics. Springer; 2012. p. 380–387.

[14] Gilmore JF, Czuchry AJ. A neural network model for route planning constraint integration. In: Neural Networks, 1992. IJCNN., International Joint Conference on. vol. 3. IEEE; 1992. p. 221–226.

[15] 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.

[16] 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.

[17] 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.

[18] Gallaire H, Minker J, Nicolas JM. Logic and databases: A deductive approach. In: Readings in Artificial Intelligence and Databases. Elsevier; 1988. p. 231–247.

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

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

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

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

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

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

[25] 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.

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

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

[28] 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.

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

[30] 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.

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

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

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

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

[35] 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.

[36] 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.

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

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

[39] 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.

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

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

[42] 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.

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

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

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

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

[47] 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.

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

[49] 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.

[50] Bengio Y, Louradour J, Collobert R, Weston J. Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning. ACM; 2009. p. 41–48.

[51] Zaremba W, Sutskever I. Learning to execute. arXiv preprint arXiv:14104615. 2014;.

[52] Feng R, Xu X, Zhou X, Wan J. A trust evaluation algorithm for wireless sensor networks based on node behaviors and ds evidence theory. Sensors. 2011;11(2):1345–1360.

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

[54] Kuntimad G, Ranganath HS. Perfect image segmentation using pulse coupled neural networks. IEEE Transactions on Neural networks. 1999;10(3):591–598.

[55] Yu B, Zhang L. Pulse-coupled neural networks for contour and motion matchings. IEEE Transactions on Neural Networks. 2004;15(5):1186–1201.

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

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

[58] 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. .

[59] 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.

[60] 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.

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

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

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

[64] 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.

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

[66] 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.

[67] 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.

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

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

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

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

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

A Dijkstra' 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**

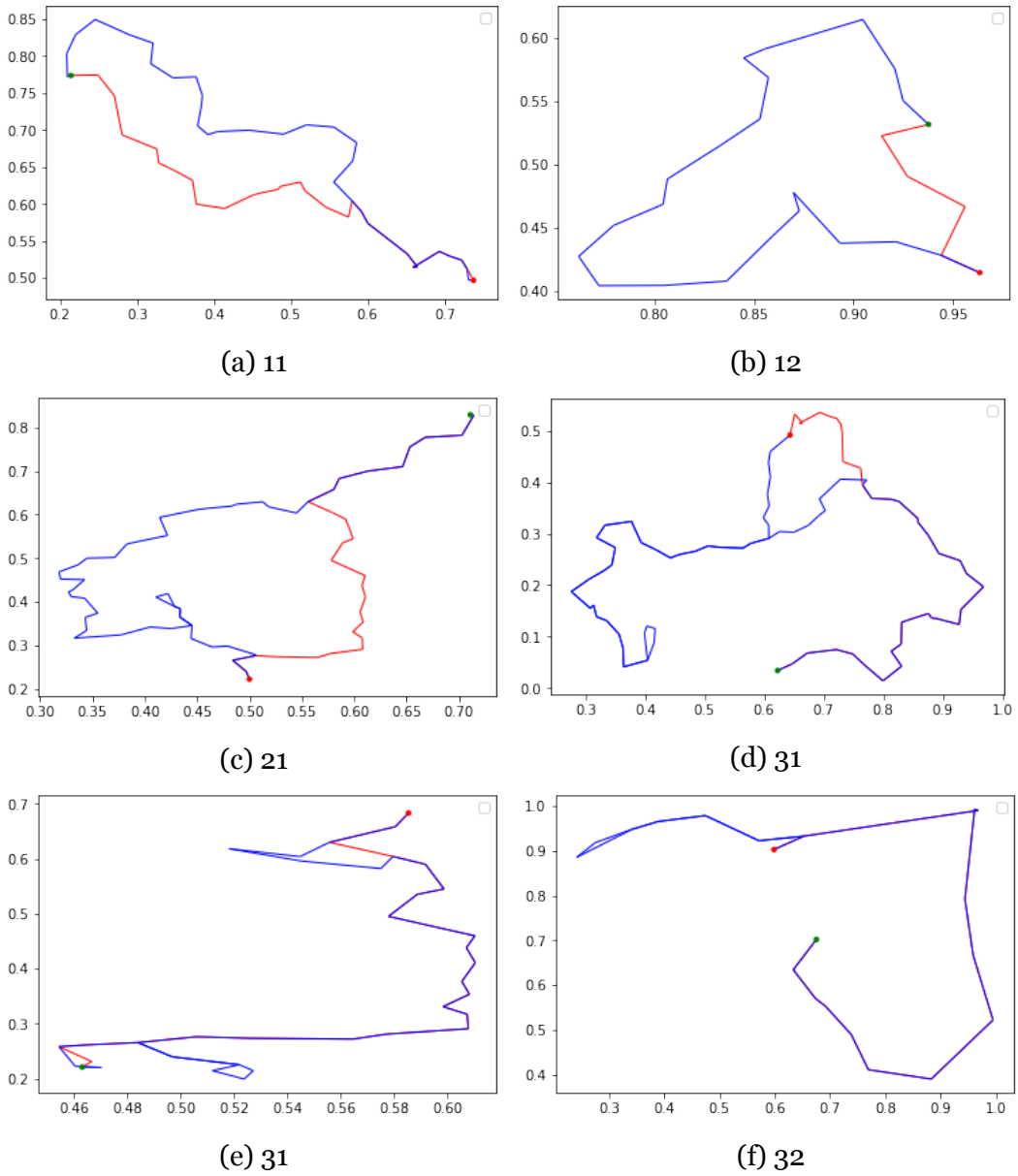
```
1  Let Set_closed denote the nodes that already been evaluated;
2  Let Set_open denote the discovered nodes that have not been evaluated;
3  Let path_map record the predecessor of each node with regard to shortest
     path;
4  Let g record g(n) with all node initialized as INFINITY;
5  Let g(Src) 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 ≥ 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 27: Plots show more examples of path comparison.

D More Examples on Road Network Paths Comparison

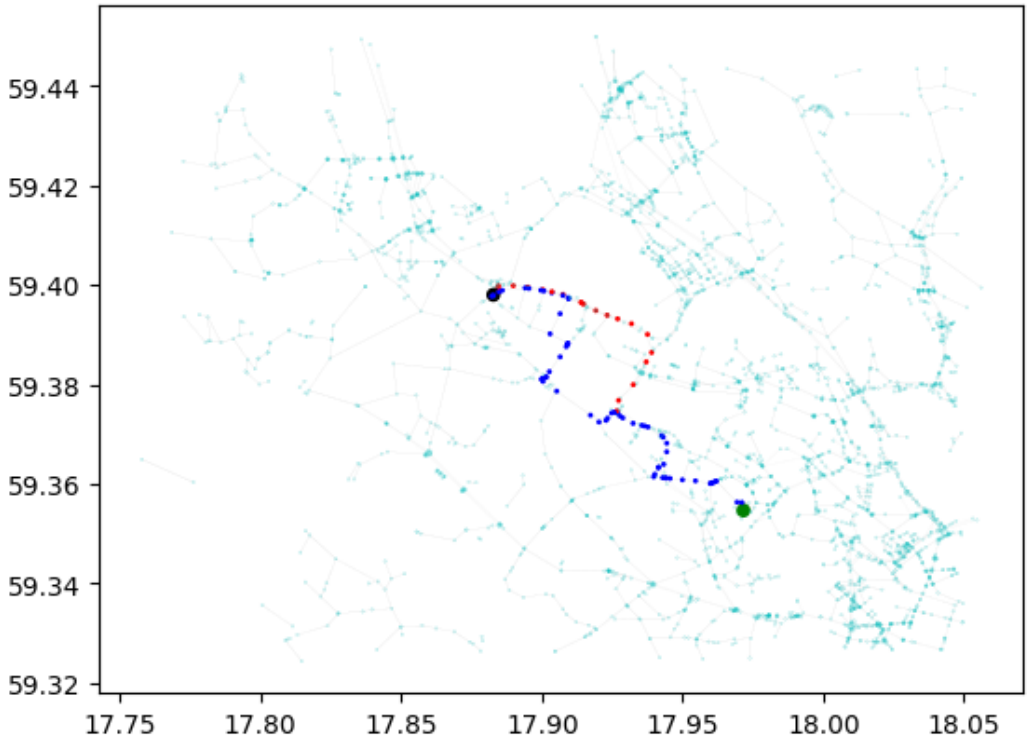


Figure 28: A plot compares the path generated by the agent (blue) and by shortest path algorithm (red).