Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Neural Distance Oracle for Road Graphs

Bachelor's Thesis

Julian Neff

neffj@inf.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Karolis Martinkus
Prof. Dr. Roger Wattenhofer

July 26, 2021

Abstract

Calculating the shortest distances on a road graph is used in a variety of today's applications. It is often necessary for the algorithm to process many queries fast and reliably for a large number of users. On huge road graphs, traditional exact distance computation algorithms, such as Dijkstra's are ill suited for such problems due to poor scaling. Fortunately, the exact distance is not always necessary. Recent work on distance prediction on road graphs used various embedding techniques combined with a multi-layer perceptron (MLP). This learning based approach is able to predict distances accurately, fast and with low space cost. In this thesis, we focused on distance prediction on MLPs with a simple model structure and see where the chances and limitations are. Unlike other recent work, we used no embedding and tried to map the coordinates of two points to the corresponding distance. We tried to reason why some models do well on some types of graphs, while they fail on others and why they sometimes fail on larger graph instances of the same type. For large road graphs, we were motivated by a new model, which was able to memorize images using a sinusoidal representation network. With this approach, we were able to predict distances with a negligible error (up to an MAE of 0.00049 and an MRE of 0.00252). This MLP reduces the complexity of the model structure and uses less space than a lookup table for a graph with 1k nodes. However, it does suffer from a long training time.

Contents

Chapter 1

Introduction

1.1 Applications

Shortest path algorithms are a well-studied problem in the field of graph theory and are used in many of the algorithms which we use every day. For instance in a social media network, to compute the closeness centrality, one uses the distance between two nodes [27] and in a trust network, the number of hops between two entities shows the level of trust [23]. As a measure, how big these graphs and how frequently used they are here some examples: In 2020 the web mapping platform "Google Maps" was being used by over 1 billion people every month [1]. In the same year, the social media platform "Facebook" recorded 2.74 billion active users [2] and in 2018, Switzerland's biggest online retailer "Digitec" showcased more than a million products in its catalog [3].

Answering distance queries on such enormous graphs poses a significant challenge in both space and time cost. However, exact distances are not always required. For instance, finding points-of-interest (POI) efficiently and fast, which has become one of the most important tasks in location-based social networks ($LBSN$), where one tries to predict the physical movement of users or launch targeted advertisement [25]. Another use case is friend recommendations in a social media network or suggested products in an online shop.

This thesis focuses on road graphs, where there are applications such as finding the optimal route. In a real world scenario, the optimal route is hard to define, because if we would try to model a street network as a graph, we would need to include external factors such as the time of the day, the speed limit of a street segment, the current weather, and so on. Sometimes a user also does not wish to take the fastest route and takes instead the route where there is the least amount of traffic or the route where there are no motorways. In this thesis, we only focus on distance prediction on road graphs. There are however some papers which focus on these kinds of dynamic networks [29, 31, 21].

1.2 Background

There are many different algorithms for calculating the exact length of the shortest path between two nodes in a graph. In a road graph with positive edge weights, the classic algorithm is **Dijkstra**. The time cost is dependent on the implemented priority queue. In an optimal case one uses *Fibonacci heaps* which reduce the time complexity to $\mathcal{O}(|E| + |V| \times log|V|)$. An improved version of Dijkstra would be the $A*$ algorithm, which takes additional heuristics into consideration. **A*** works at least as quickly as Dijkstra, but they both still have the same time complexity [12].

There are also algorithms which work on graphs with negative weights (e.g. **Bellman-Ford**) but these are not relevant when looking at road graphs, as all weights are always non negative.

Instead of calculating the shortest distance each time one could also store the distance between any pair of nodes in a lookup table, where a processing of a query could be done in $\mathcal{O}(1)$. Unfortunately this has the drawback of having a space cost of $\mathcal{O}(|V|^2)$, which is not feasible when we have large real world road graphs.

All these algorithms have the drawback that their time and/or space cost does not scale well with the graph size. Especially in a road graph with millions of nodes and modern applications, which need to be able to address multiple queries fast at each moment in time. Therefore, if the algorithm cannot guarantee this, then it is not feasible to use it.

1.3 Our contributions

1. We looked at simple neural networks and their abilities and limitations to memorize graphs. The focus was laid on how these models scale with the graph size.
2. For large scale road graphs, where it is essential that fine details are memorized, we propose to use sinusoidal activation functions.
3. We evaluate the different methods to store path distances and give suggestions when each methods should be used.

Chapter 2

Related Work

2.1 Landmark Labels

Most distance approximation algorithms take a landmark based approach, where a subset of k ($k << n$) nodes are chosen as landmarks L. For every vertex v_i the distance to all other landmarks is computed in a lookup table. This reduces the space cost to $\mathcal{O}(k \times n)$. To approximate the distance between two nodes v_i and v_j the landmark with the shortest distance is chosen and the distance is calculated as the sum of the nodes to this landmark [16]:

$$dist(v_1, v_2) \approx min\{D(v_i, l) + D(v_j, l) \mid l \in L\} \tag{2.1}$$

This can be done in $\mathcal{O}(|L|)$. Choosing the number of landmarks is a trade-off between space and accuracy. The accuracy of the approximation also heavily depends on the selected landmarks. Finding optimal landmarks is NP-Hard [18], therefore some heuristics were suggested by Zhao [32] to measure the *centrality* of the vertices and from that choose the landmarks:

1. Degree (Number of connections of a node) centrality: The assumption is that a node with a high degree will probably be traversed a lot.
2. Betweenness (Number of shortest paths passing through the edge): A node with a high betweenness will be traversed a lot. This is computationally very expensive.
3. Closeness (Average distance to all nodes): A node with a small closeness is likely to be at the center of the graph.

One drawback of this approach is that one has to be cautious that the nodes are still evenly distributed. It is assumed that graphs with a hierarchical structure do well with degree centrality guided landmarks [32]. Takes and Kosters [24] have shown that betweenness outperforms closeness and propose a selection strategy that balances between coverage and centrality. Zhao [32] goes further into the different strategies. There are also labelings which ensure that the distances can

be recovered perfectly, but then the label size becomes larger. At best, the labeling is $\mathbb{O}(log\ n)$ times larger than the best possible labeling that still ensures perfect distance recovery [5].

Theoretical results have further shown some insights in the relationship between landmark size and accuracy.

Let $\widetilde{d}$ be the approximation of the effective distance d, then the **approximation ratio / Stretch** is defined as follow:

For some $\alpha \geq 1$ and $\beta \geq 0$ the following holds for all pairs of nodes

$$d(v_i, v_j) \leq \widetilde{d}(v_i, v_j) \leq \alpha \times d(v_i, v_j) + \beta \tag{2.2}$$

From [19]: On undirected graphs Thorup and Zwick [26] have shown that for any algorithm with approximation error of $\alpha \leq 2c + 1$ where $c \in \mathbb{N}^+$ must use at least $\Omega(n^{1+\frac{1}{c}})$ space. Furthermore they have provided a model which uses $\mathbb{O}(c \times n^{1+\frac{1}{n}})$ space and $\mathcal{O}(c \times m \times n^{\frac{1}{c}})$ time cost to obtain an approximation ratio of $\alpha = 2c - 1$ and a $\mathcal{O}(c)$ query time. Chechik [7] improves it to $\mathcal{O}(n^{1+\frac{1}{c}})$ space cost and $\mathcal{O}(1)$ query time with an increased $\mathcal{O}(n^2 + mn^{\frac{1}{2}})$ processing time, where m is the number of edges. These studies are only of theoretical interest and do not provide any empirical data.

2.2 Graph Embeddings

Graph embedding techniques can be used to convert high-dimensional sparse graphs to low dimensional dense vector spaces while still preserving graph structure properties. Embedding techniques are in general not designed to predict vertex distances but they can be modified to do it. Moreover the most recent work that we found [20, 19, 32] in distance prediction makes use of embeddings in some form. Most graph embedding techniques in the literature can be broken down into three categories.

Matrix factorization-based methods

Matrix factorization-based methods construct a high-order proximity matrix based on transition probabilities and factorize it to obtain the node embeddings [30]. These methods cannot easily scale up to large networks [30]. To show this we quickly go over two examples for Matrix factorization-based methods: *Laplacian Eigenmaps* [6] embeds vertices that share an edge with a large weight to be close in the latent space. It tries to minimize $\left\|Z_i - Z_j\right\|^2 W_{ij}$ where Z is the embedd matrix and W is a penalty in the embedd space [32]. A further example is the

Locally Linear Embedding, where the embedding of a node is the sum of vectors of its neighbors and is defined as $Z_i = Y_{ij} Z_j$, where Y is the adjacency matrix [32].

Both of these examples can be solved as an eigenvalue problem and have a computational complexity of $\mathcal{O}(|E| k^2)$ where k is the dimension size of the embedding [32] and are therefore not feasible to apply them in large scale road graphs.

Random walk-based methods

The random walk technique samples the graph as a discrete distribution and embeds the sample into vectors [32]. Papers [20, 19, 32], which also dealt with distance prediction made use of *node2vec* [11], which focuses on the neighborhood of the nodes. The idea is that nodes with similar neighborhoods tend to have similar representations. In [20] *node2vec* simply runs random walks on the graph and feeds them into *word2vec* [17], a natural language processing technique for sentences and words. Rizi et al. [20] suggested an embedding dimensionality of $k = 128$ to be optimal. In comparison to matrix factorization techniques, *node2vec* has the lower computational cost and the results are still precise [11].

Neural network-based methods

Neural network-based methods are generated from training an encoder and a decoder, and trying to minimize the error from mapping the input into the latent space and then back to the original format [9]. One method would be the *Structural Deep Network Embedding* which takes an adjacency matrix of a graph and adapts the loss function to preserve both the first and second order proximities [28]. Summarized by Zhao [32]:

> **First order proximity:** When nodes are mapped far away in the embedding space a penalty is added
>
> **Second order proximity:** Errors in the output which correspond to nonzero elements in the input are more penalized

In Summary: Graph embedding techniques offer linear space cost and fast query processing time but are challenging to set up while still preserving distance information.

2.3 Sinusoidal representation networks

Instead of embedding the distances, another approach could be to compress the lookup table with a *multi-layer perceptron (MLP)*. MLPs in combination

with some form of graph embedding have already been used in various papers [11, 19, 32, 8, 33] on road graphs and networks to predict the shortest distances, travel time or path finding with great success. They did this by sampling distances from paths in the graph and learning others from this.

Implicit neural representation are signal representation that parameterize implicitly defined functions. These signal representations are usually combined with a continuous parametrization, to make it more memory efficient and also being differentiable, with the hope that one can solve inverse problems, such as differential equations.

Over the last year signal representations parameterized by neural networks have emerged as a powerful paradigm and have been well studied. Many of the recent work focuses on the popular $ReLU$ activation function. Sitzmann et al. [22] believe that these architecture lack the capacity to model fine details and propose periodic activation functions for implicit neural representations, which are called *sinusoidal representation networks* or *SIREN*.

They have shown that periodic activation functions are ideally suited for representing complex natural signals and their derivatives using implicit neural representations [22]. Examples for such natural signals are images, audio and video.

Recently Dupont et al. [9], suggested a model which made use of *SIREN*, and has shown to accurately map 2D pixel coordinates to a corresponding color. They did this with a very deep model which mapped 2D coordinates to RGB color values. Additionally they did a lot of iterations over the model and used low learning rate to train it. Their model outperformed JPEG images in low bit rates.

Having seen that it is possible to compress images in this way, we hope to accurately map the two 2D coordinates to a distance.

Chapter 3

Distance Approximation

3.1 Problem formulation

We consider a road graph $G = (V, E)$, where V ($|V| = n$) denotes the set of nodes, which represent road intersections and dead ends in a real world scenario. Furthermore let E ($|E| = m$) denote the set of edges, where each edge represents a street segment. Each node $v_i \in V$ has a pair of coordinates and each edge $e_i \in E$ has a weight representing the length of the segment. Let $d(v_i, v_j)$ be the exact distance function between two nodes, our problem now is to find a function $\hat{d}(v_i, v_j)$ such that the difference to the exact distance function is negligible and the time and space complexity scale slow with the graph size.

3.2 Preliminaries

3.2.1 Multi Layer Perceptrons / Neural Networks

Multi Layer Perceptrons / Neural Networks is a concept that dates back to the late 1950s which has been inspired by the neural system of the human brain. In the last couple of years, with the rise of faster GPU's and cheaper memory, neural networks have emerged into one of the most popular machine learning model. Applications for neural network are in *image classification*, *natural language processing*, *autonomous cars* and so on.

A neural network consists of input and output neurons, which are connected through a series of hidden layers. Each connection is weighted and each neuron can have some form of activation function, where the inputs are processed. The output is calculated through a forward-propagation where the input neurons are activated and flow through the network until the output neurons are reached. Then the weights can be adapted to minimize the error using back-propagation, where the output neurons are activated and the gradients are calculated from the end of the model to the front.

3.2.2 Implicit Neural Representations with Periodic Activation Functions

The following section is a summary of the papers from Sitzmann et al. [22] and Dupont et al. [9]. Implicit neural representations are neural networks that learn how to parameterize implicitly defined functions. A continuous parametrization offers the potential to be more memory efficient, allowing the model to learn finer details. Additionally, being differentiable allows the model to calculate higher order derivatives analytically. Combining these two may offer a new toolbox for solving inverse problems, such as differential equations [22].

Most research in the last years has made use of the *ReLU (Rectified Linear Unit* activation function, which is defined as follows:

$$f(x) = max(0, x) \tag{3.1}$$

ReLU is piecewise linear, which means that the second derivative is zero everywhere. This can imply why *ReLU* fails to learn fine details. Also activation functions that do not suffer from this problem and have non-zero second derivatives such as *tanh* or *softmax* are often not well behaved [22]. Following these assumptions, a periodic activation functions for implicit neural representations was suggested in [22], which are able to learn finer details of the signals and their derivatives.

SIREN is a simple neural network architecture for implicit neural network architecture that enables accurate representation of natural signals called *SIREN*, such as images audio and video in a deep learning model [22]. It uses *sine* as a periodic activation function which is defined as follows:

$$\Phi(x) = \mathbf{W}_n(\phi_{n-1} \circ \phi_{n-2} \circ ... \circ \phi_0)(x) + \mathbf{b}_n, \quad x_i \mapsto \phi(x_i) = \sin(\mathbf{W}_i x_i + \mathbf{b}_i) \tag{3.2}$$

Where $\phi_i : \mathbb{R}^{M_i} \mapsto \mathbb{R}^{N_i}$ is the i-th layer of the network, $\mathbf{W} \in \mathbb{R}^{N_i \times M_i}$ is the weight matrix and $\in \mathbb{R}^{N_i}$ is the bias.

In comparison to other activation functions such as *ReLU* the derivative of *sine* is also a *SIREN*. It was also shown that when *SIREN* is supervised using a constraint $\mathbf{C}_m$ involving the derivatives of ϕ, then ϕ remains well behaved [22].

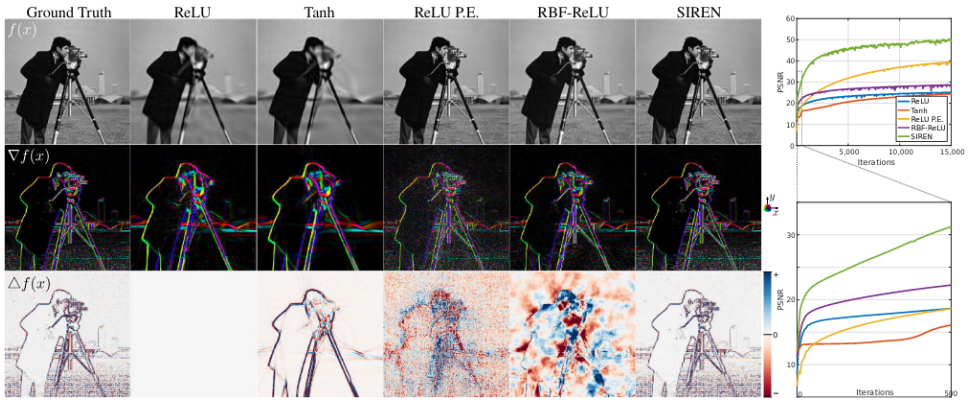

Figure 3.1: Direct copy from [22]: Comparison of different implicit network architectures fitting a ground truth image (top left).The representation is only supervised on the target image but we also show first- and second-orderderivatives of the function fit in rows 2 and 3, respectively.

For the model itself, the weight initializations were chosen carefully, such that the final output in the last layer is independent of the depth of the model. Sitzmann et al. [22] did this by preserving the distribution of activation functions throughout the whole network.

Sitzmann et al. [22] showed that periodic activation functions using implicit neural activations are ideally suited to represent complex natural signals and their derivatives. It also is able to solve several boundary value problems robustly.

Dupont et al. [9] have recently proposed a new model *COIN* to compress images, which makes use of the *SIREN* model. Their target was to minimize the following equation using the *mean squared error* with the least number of parameters possible:

$$\min_{\theta} \sum_{x,y} \left\| f_\theta(x,y) - I[x,y] \right\|_2^2 \tag{3.3}$$

Here, f_θ is the mapping from the neural network that maps coordinates to RGB values of the image and $I[x,y]$ is the correct RGB value at position (x,y) of the image.

In [9] it was shown that the *COIN* model outperforms JPEG photos in low bit rates. The main limitation of this model is that the encoding for each image takes long and is unique to this picture. What's interesting is that they used very deep neural networks, in some cases up to 13 (very small) layers. Another interesting take away is that they overfitted the model with over 50'000 epochs and used a very low learning rate of $2e-4$.

3.3 Graphs

To test our models, we did various tests over different kind of graphs. Here we just briefly go over the definitions of the graphs and explain why they are of interest to us.

3.3.1 Claw Graph

A claw graph is the *complete bipartite graph* $K_{1,3}$, e.g. one central node which is connected to three outer nodes. It is a simple graph that cannot be embedded in the Euclidean space such that the distances between all nodes in the space would equal the distances in the graph [15].

We are interested in seeing how small one can scale the model and still achieve good results as well as how much training is necessary.

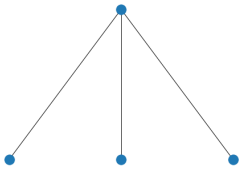

Figure 3.2: A claw graph

3.3.2 Grid Graph

A $m \times n$ two-dimensional grid graph is a graph where the nodes are aligned in a grid and each node is connected with its closest neighbors.

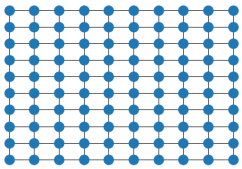

Figure 3.3: A 10x10 grid graph

The distance between two nodes can be calculated with the *L1 norm / Manhattan distance*. Let $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$ be some arbitrary point on the grid. Then the distance between them can be calculated as:

$$d(p_i, p_j) = |x_i - x_j| + |y_i - y_j| \tag{3.4}$$

This makes it trivial to predict distances on a grid. However, we are interested in seeing how models, which can predict distances really well on the grid graph behave on manipulated grid graphs.

Grid with random dropped nodes

To simulate real world road graphs, we used a grid graph and deleted a small proportion of nodes in it, but still made sure that the graph is connected.

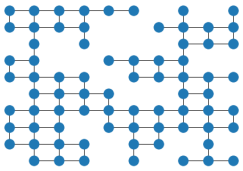

Figure 3.4: A grid graph where a quarter of all nodes have been randomly removed

Grid with random edge weights

Another approach to simulate road graphs was to give each edge a non-deterministic weight in the range of $[1 - x, 1 + x]$ for some $x \in [0, 1]$. This gives a nice comparison to the grid manipulation where we exclude nodes and we will learn more about which manipulation has the bigger influence on the model.

3.3.3 Binary Tree

A binary tree is a hierarchical data structure where there is one central root node and layers of children, where each node has at most two children. The distance between two nodes can be calculated if the lowest common ancestor is known. Unfortunately, this is only possible with a tree traversal algorithm, which has time complexity $\mathcal{O}(n)$. Graph types with hierarchical structures have

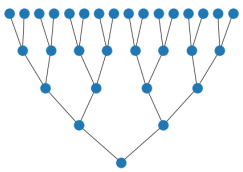

Figure 3.5: A complete binary tree of height 4

also been suggested to do well on landmarks by Zhao [32], if one uses centrality guided landmarks. The hope is that neural networks also utilize the hierarchical properties of the graph.

3.3.4 Random geometric graph

In our experiments we also made tests on random geometric graphs, created with the python package *NetworkX*. This graph is created by placing nodes uniformly at random and then placing edges if the distance between two nodes is smaller than a given range.

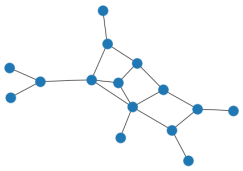

Figure 3.6: A random geometric graph

We tested on these kinds of graphs because we assumed that if a model is able to completely memorize a random geometric graph, where the only constraint are the proportion of nodes to the edges, then it should also be able to memorize a graph with more constraints, like road graphs.

3.3.5 Road Graph

A road graph is an attempt to model a street network. It isn't defined in some form but there are characteristics to it. From our findings the number of edges scales linearly with the number of nodes, usually with a factor in the range of two to five [4]. Additionally overlaps between edges are rare and appear only in edge cases, e.g. a tunnel, this gives the model some simplicity. However, apart from this road graphs can be very varied as they are influenced by external factors such as rivers and cliffs.

Figure 3.7: A road graph of Zürich made with the python package OSMNX

Chapter 4

Model

4.1 Model Structure

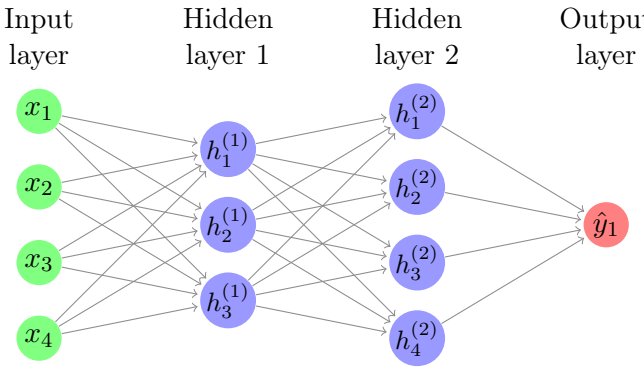

Figure 4.1: An example structure of our model with two hidden layers

For all tests, our model will have an input size of 4 nodes representing the coordinates of two points $p_1 = (x_1, x_2)$ and $p_2 = (x_3, x_4)$. The model will then have multiple hidden layers of various sizes which are always fully connected and at the last layer there is one output neuron.

We deliberately trained the model over the whole dataset (*APSP* table) with a high number of epochs, similar to Dupont et al. in [9]. Additionally we fixed the batch size to the number of nodes, as we did not see any indication that any other batch size improves the results from our tests.

4.2 Settings

4.2.1 Weight Initializations

Weight initialization of a model is the procedure for how the weights of the neurons are first initialized. The initializations are meant to preserve the norm of the gradients, which could speed up the convergence.

Kaiming

Fills the input according to Kaiming et al. [13] using a uniform distribution $U(-bound, bound)$ where *bound* is defined as:

$$bound = gain \times \sqrt{\frac{3}{fan_{mode}}} \tag{4.1}$$

fan_{mode} is a setting where one can either choose to preserve the magnitude of the weights in forward passes or backward passes. *gain* is a scaling factor. This function is meant to work well with the $ReLU$ function. It is an improvement over 4.2.1.

Xavier / Glorot

Fills the input according to Xavier et al. [10] using a uniform distribution $U(-a, a)$ where a is defined as:

$$a = gain \times \sqrt{\frac{6}{fan_{in} + fan_{out}}} \tag{4.2}$$

Here, *gain* is a scaling factor and fan_{in} (fan_{out}) refers to the maximum number of input (output) neurons a system can accept. The aim is to have the same variance for the activation functions on each layer. This should help to reduce the chance that the gradient explodes or vanishes.

4.2.2 Activation functions

The *activation function* of a node is a function applied to the input of that node. Having activation functions allows the model to learn nonlinear functions.

ReLU

Is the most popular activation function. It is a rectified linear unit function:

$$ReLU(x) = max(0, x) \tag{4.3}$$

GELU

Gaussian Error Linear Units were first proposed by Hendrycks and Gimpel [14] and is defined like follows:

$$GELU(x) = xP(X \leq x) = x\phi(x) = x \cdot \frac{1}{2}[1 + erf(x/\sqrt{2}] \tag{4.4}$$

where *erf* is the error function. $GELU$ can be approximated with:

$$GELU(x) \approx 0.5x(1 + \tanh\,[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)] \tag{4.5}$$

4.2.3 Adaptive learning rate

An *adaptive learning rate* is an optimization of the gradient descent methods, where the learning rate changes dynamically during training.

Multiplicative decrease

Multiplicative decrease decays the learning rate by a given factor after a number of given number of epochs.

Reduce learning rate on plateau

Reduces the learning rate once the metric has stopped improving after a certain number of rounds (i.e., a plateau has been reached).

4.2.4 Loss Functions

The loss functions describe how the effective loss is calculated after each batch of inputs. Here we just briefly go over the loss functions that were used in the tests. $\hat{y}$ represents the output of the model while y represents the exact distance. n corresponds to the number of samples in a batch.

Mean Squared Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.6}$$

L1 / Mean Absolute Error

$$L1 = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.7}$$

Mean Squared Relative Error

$$MSE_{norm} = \frac{1}{n}\sum_{i=1}^{n}(\frac{y_i - \hat{y}_i}{y_i})^2 \tag{4.8}$$

Mean Absolute Relative Error

$$L1_{norm} = \frac{1}{n}\sum_{i=1}^{n}|\frac{y_i - \hat{y}_i}{y_i}| \tag{4.9}$$

Reverse Huber loss

This loss function was suggested by Qi et al. [19].

$$ReverseHuber = \frac{1}{n}\sum_{i=1}^{n} f_\sigma(y - \hat{y}) \tag{4.10}$$

where f_σ is defined as follows in our experiments:

$$f_\sigma(x) = \begin{cases} \delta|x|, & \text{if } |x| \leq \delta \\ \frac{1}{2}(x^2 + \delta^2), & \text{otherwise} \end{cases}$$

Where $\delta = 0.2$

In all experiments, we used a batch size of size n. We tested this rigorously on various graphs and models and concluded that this batch size does well for most of them.

Chapter 5

Graph Memorization

5.0.1 Evaluation criteria

The tests were evaluated by their *Mean absolute error (MAE)* and *Mean relative error (MRE)*. Additionally each test was run five times to get the variance of the results. We categorized the results the following way:

1. $MAE < 1\%, MRE < 5\%$: Was considered good
2. $MAE < 0.1\%, MRE < 1\%$: Was considered perfect

Additionally, we tried to squeeze the *layer sizes* and the number of *epochs* in the model as much as possible, because a model is not useful if it is computationally too expensive.

5.1 Distance prediction with simple neural networks

Most papers which have dealt with the same problem using a neural network have usually combined it with some form of landmark and/or embedding approach [20, 19, 32, 8]. We take a step back from the approach these papers took and look at simple models on simple graphs. Using this strategy, we hope to get some insights in the following questions:

1. What are the possibilities for a simple neural network? What are the limitations?
2. Why do models do well on specific graphs but fail on others?
3. How do models change when we scale up the graph size?

5.1.1 Claw Graph

There were 19263 tests done in total over a wide range of settings on the claw graph from 3.3.1. The model weights were initialized with *kaiming* and the activation function was *GELU* for all hidden layers. We used a multiplicative decreasing learning rate where the learning rate decreased by a factor of 10 after 90% of epochs. Most of the configurations were able to memorize the graph with little computational cost, we therefore only provide one example of such a setting.

Loss F.	**Learning R.**	**Epochs**	**Structure**	**MAE**	**MRE**
MSE	0.1	440	[4,4,4]	0.00001	0.00001
MSE	0.1	80	[3,1,1]	0.00079	0.00119

Table 5.1: Some of the model configurations that allow for perfect memorization of the claw graph

Using these settings, we were able to memorize the graph. It is encouraging that one is able to memorize the claw graph with such a small model.

5.1.2 Grid Graph

There were 3670 tests done on a 10×10 grid and 186 on a 25×25 grid defined in 3.3.2. We used the same loss functions as described in the settings section and a learning rate in the range of $[0.1, 0.0001]$.

The models used a multiplicative decreasing learning rate where the learning rate decreased by a factor of 10 after 90% of epochs. For the tests we used a batch size of n (i.e. the number of nodes).

With the following settings, we were able to memorize a 10×10 grid, resp. 25×25 grid.

Graph	**Loss F.**	**Learning R.**	**Epochs**	**Structure**	**MAE**	**MRE**
10x10	MSE	0.04	80	[20,20,20]	0.00095	0.00352
25x25	L1	0.1	20	[40,40]	0.003	0.011

Table 5.2: Different settings for a model on the grid graph

From our results, we see that a relatively simple model can memorize a grid graph with little computational power needed. However, in both of these cases we have more parameters in the model than there are distances.

Manipulated grid graphs

Here we still use a grid graph but modify it dropping random nodes or giving the edges random weights as described in 3.3.2.

All tests were done on a modified 10×10 grid. We used models, which were able to memorize a normal grid given specific settings. Each model had one *ReLU* 4.2.2 activation function on the last layer. The different model structures are described below:

Model	Layer 1	Layer 2	Layer 3
M1	20	20	-
M2	10	10	10
M3	15	15	15
M3	50	50	50

Table 5.3: Different Model structures used

The results are the average results over five runs. We used a learning rate of 0.01 and did not modify it during training. Each configuration was trained for 10 epochs.

Manipulation MAE	M1	M2	M3	M4
Random weight [0.5 - 1.5]	2.2%	2.4%	2.3%	2.4%
Random weight [0.6 - 1.4]	1.7%	3.4%	1.9%	2.0%
Random weight [0.7 - 1.3]	1.5%	1.6%	1.4%	1.6%
Random weight [0.8 - 1.2]	1.1%	1.1%	1.1%	1.2%
Random weight [0.9 - 1.1]	0.5%	0.5%	0.4%	0.6%
Normal Grid	0.0%	0.0%	0.0%	0.3%
Nodes dropped [5%]	0.6%	0.6%	0.6%	0.8%
Nodes dropped [10%]	1.0%	2.7%	1.0%	1.3%
Nodes dropped [15%]	2.5%	2.5%	2.5%	2.8%
Nodes dropped [20%]	5.1%	5.3%	5.2%	5.7%
Nodes dropped [25%]	5.5%	5.9%	5.7%	6.7%

Table 5.4: Tests results over each model structure (MAE)

In practice, a road graph has no structure one can use as an assumption. A similarity that a road graph and a manipulated grid graph have is that there are no (or negligible part) of overlapping edges. Therefore these results might give an insight on how much the different manipulations impact a model.

Manipulation MRE	M1	M2	M3	M4
Random weight [0.5 - 1.5]	8.5%	8.7%	8.5%	8.5%
Random weight [0.6 - 1.4]	6.3%	12.8%	6.9%	7.1%
Random weight [0.7 - 1.3]	5.1%	5.4%	5.1%	5.5%
Random weight [0.8 - 1.2]	3.9%	3.9%	3.8%	4.1%
Random weight [0.9 - 1.1]	1.7%	1.7%	1.6%	2.1%
Normal Grid	0.1%	0.1%	0.2%	0.9%
Nodes dropped [5%]	1.5%	1.4%	1.4%	2.4%
Nodes dropped [10%]	2.8%	9.6%	2.6%	3.5%
Nodes dropped [15%]	6.8%	6.6%	6.7%	7.0%
Nodes dropped [20%]	12.9%	13.2%	12.8%	12.3%
Nodes dropped [25%]	18.3%	18.4%	19.4%	14.9%

Table 5.5: Test results over each model structure (MRE)

5.1.3 Binary Tree

Zhao [32] suggested that distance prediction with landmark labeling does well on graph with hierarchical structures. This motivates us to see if a neural network also profits from a hierarchical structure, by doing tests on a binary tree from section 3.3.3.

There were 67520 configurations done on a tree with 7 nodes, 94482 on a 15 node tree and 11411 on a 31 node tree.

The models used a multiplicative decreasing learning rate where the learning rate decreased by a factor of 10 after 90% of epochs. We tested with all loss functions as described in the settings section. The learning rate was in the range of $[0.5, 0.00001]$ and the number of epochs in $[10, 10000]$. The weights were initialized with the standard method and the activation function was $ReLU$ for each layer.

Binary Tree: n = 7

From the plots in Fig:5.1 and Fig:5.2 of the layer sizes, one can see that the models can only fit the tree well with a very high layer size and a very high amount of epochs. All configurations that did well used the loss function *MSE* 4.2.4.

Binary Tree: n = 15

Compared to the tree with 7 nodes, the results visualized in fig:5.3 and fig:5.4 have become worse. The layer sizes increased by 50% and the loss functions that

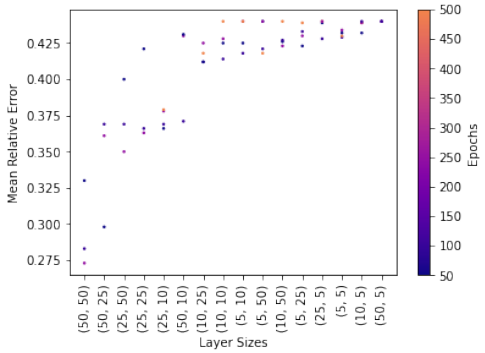

Figure 5.1: Binary Tree (n = 7): Comparison of layer sizes (of best 0.5% of runs

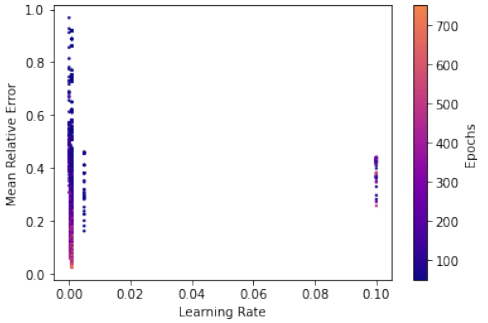

Figure 5.2: Binary Tree (n = 7): Comparison of Learning Rates (of best 5% of runs)

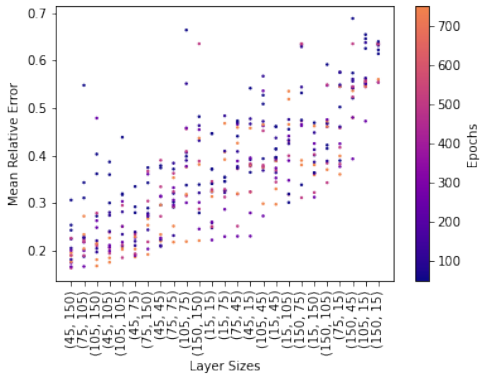

Figure 5.3: Binary Tree (n = 15): Comparison of layer sizes (of best 0.5% of runs)

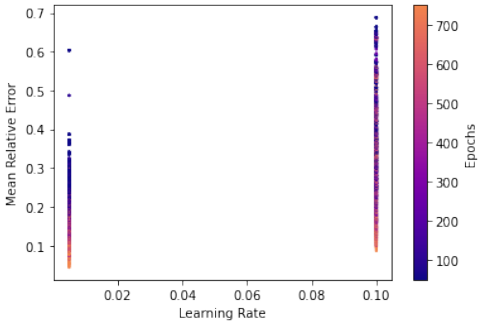

Figure 5.4: Binary Tree (n = 15): Comparison of Learning Rates (of best 5% of runs)

did well are now *L1* and *Reverse Huber* instead of *MSE*.

Binary Tree: n = 31

Comparing it with the smaller trees in fig:5.6 and fig:5.5, the results have gotten worse again and we were not able to find a setting for which the model did well on this graph (The best test had a *MAE* of 0.032 and a *MRE* of 0.079, which is higher than our minimum baseline of a $MRE < 5\%$).

We see that layer size change contributes the most to model error, which leads us to believe that the number of parameters is the driving factor behind model accuracy.

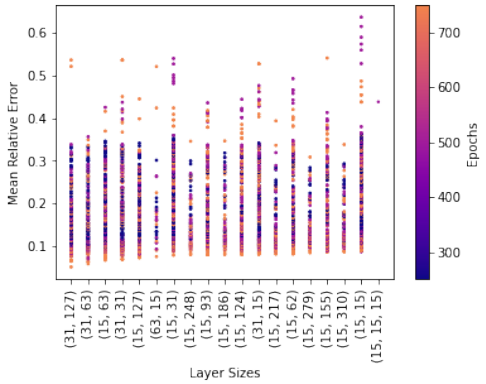

Figure 5.5: Binary Tree (n = 31): Comparison of layer sizes (all experiments)

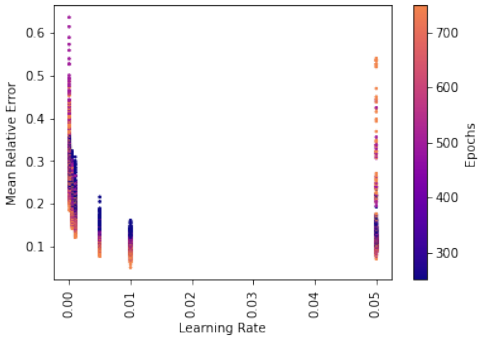

Figure 5.6: Binary Tree (n = 31): Comparison of Learning Rates (all experiments)

5.1.4 Random geometric graph

For the tests on the random geometric graph from section 3.3.4, we used the *Xavier* initialization and $GELU$ as the activation function on each layer, because from the initial testing it seemed to do better. We used all 5 different loss functions as described in the settings in section 4.2.4 and the learning rate ranged from $[0.0001, 0.3]$.

The number of tests for each graph can be seen in the table below.

7 nodes	15 nodes	30 nodes	45 nodes	60 nodes	75 nodes
6336	49719	3484	1030	217	819

With enough fine tuning, we were able to memorize the graphs up to 75 nodes. However, like in the binary tree graph, this comes with a cost of large hidden layers and many epochs. From the plot in Fig: 5.7, we assumed that the number of parameters of a model scales with the number of nodes in a graph. Fitting a quadratic curve to the models that achieve $< 5\% MRE$ and have the lowest parameter count of each graph size with the least squares methods gives us a function of $f_{5\%}(x) = 35x^2$. Looking at Figure 5.8, the fitted function would be $f_{1\%}(x) = 20.8x^2$.

If either function would be true, then this would mean that at one point the model would become bigger than the lookup table of a graph, making the model completely useless. We conclude that having a bigger model certainly helps but is no guarantee for memorizing a road graph. Our models lack the ability to memorize fine details and being able to accurately represent the derivative of the graph. This brings us to our next test, where we try to adapt our model, such

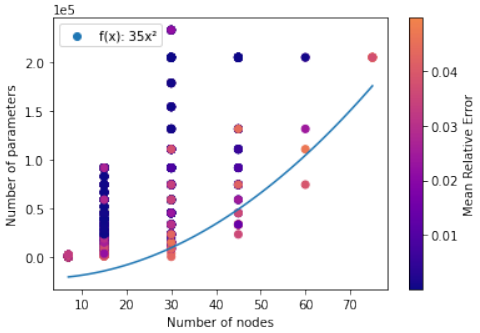

Figure 5.7: Random geometric graphs: Number of parameters of models which were able to memorize graphs

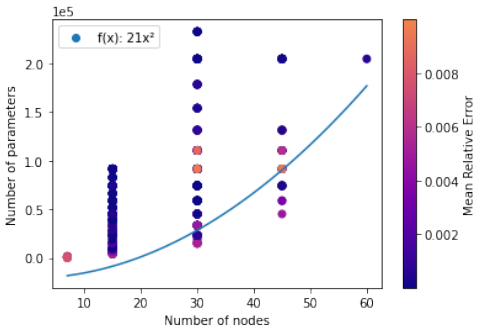

Figure 5.8: Random geometric graphs: Number of parameters of models which were able to perfectly memorize graphs

that it doesn't suffer from these drawbacks.

5.2 Distance prediction with the SIREN model

Motivated by Dupont et al. [9], we decided to use sinusoidal activation functions.

5.2.1 Model initialization

For our tests, we used different *SIREN* models with a layer size in the range of $[100, 1000]$. Additionally we tested out models with two hidden layers, similar to our tests on simple graphs and very deep neural networks similar to the model suggested by [9] with up to 12 hidden layers. The weights were initialized the same way as in [22], but we used $w_0 = 1, w = 1$ for the weight parameters.

For our graph we used a real road graph of Zürich. A visualization can be found in Figure 3.7. The graph consists of 1003 nodes and 2142 edges. The distances were normalized over the longest calculated distance, leading to an average path length of 0.379 and a variance of 0.028.

There were 522 tests done in total over this road graph.

5.2.2 Results

Layer Sizes & Number of Layers

From our findings, there was no loss of function, which outperformed others. The learning rate is similar to the work in [22, 9] in the range of $[1e-2, 1e-4]$. All

Loss F. (LR)	Epochs	Nr. Layers	Layer Sizes	MAE	MRE
L1 (0.001)	250	12	250	**0.00466**	**0.01999**
MSE (0.001)	250	11	250	**0.00383**	**0.0216**
Reverse Huber (0.003)	500	10	500	**0.00049**	**0.00252**

Table 5.6: Some settings that did well to predict the distances on the Zürich road graph

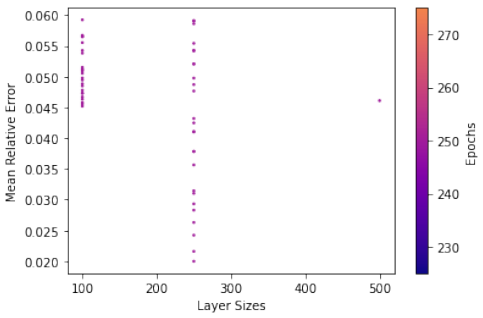

Figure 5.9: Layer sizes of best 10% of runs on Zürich road graph

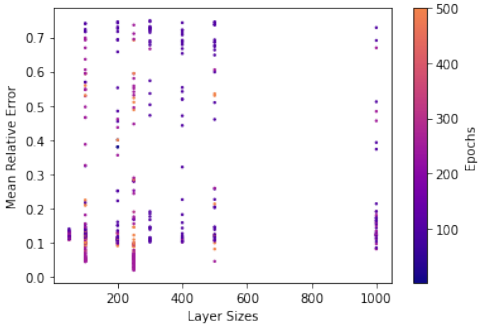

Figure 5.10: Layer sizes of best 90% of runs on Zürich road graph

tests were done 3 times each and the variance is close to zero (*e.g.*0.00001) for the tests that did well (except the test where we achieved the best results, where we had one outlier). The results show that the results do not scale with the model size like in Figure 5.7 and Figure 5.8. A reasonable model size seems to be essential but increasing it gives no guarantees for better results.

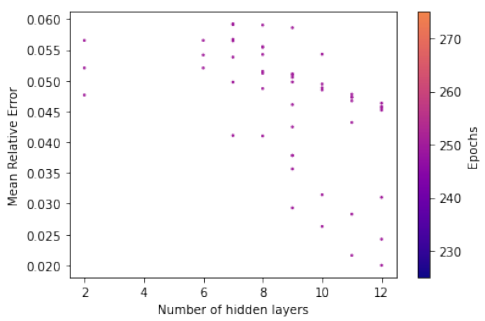

Figure 5.11: Number of layers of best 10% of runs on Zürich road graph

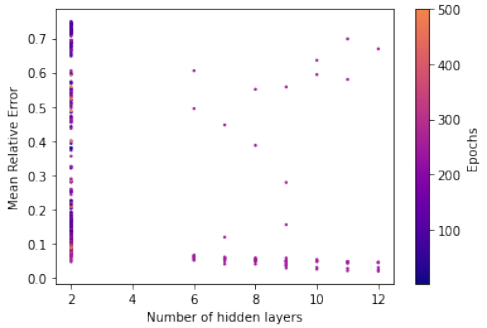

Figure 5.12: Number of layers for best 90% of runs on Zürich road graph

5.3 Comparison of memory consumption

Here we will have a look at the different methods that we have looked at in this thesis and look at their effective memory consumption.

Lookup Table

A lookup table where we store the distances has $|V|^2$ entries. Assuming that each entry uses 4 bytes of space leads to the following relationship between V and the needed space:

$$\frac{|V|^2 * 4}{10^6} = \text{Space of lookup table in MB's} \tag{5.1}$$

Which yields the following effective values: There are options to compress the

Number of nodes	100	500	1'000	2'500	5'000	7'500	10'000	25'000	50'000	75'000	100'000
MB's needed	0,04	1	4	25	100	225	400	2'500	10'000	22'500	40'000

table and then use effectively less space. The whole purpose of these values is to give a feeling on when it makes sense to start using a lookup table or switch to a neural network.

Considering that our model, for which we were able to memorize the road graph of Zurich with a mean relative error of around 2%, only needs 2.8 MB of memory, it already makes sense to use a neural network over a lookup table.

Neural Network

This table shows that even very large and deep neural networks are very memory efficient to store and that they scale very slowly with the number of parameters. Knowing that we were able to store the road graph of Zurich in a model with 12

LS \HL	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
100	0.004	0.045	0.086	0.127	0.169	0.201	0.251	0.292	0.333	0.374
200	0.007	0.168	0.33	0.491	0.652	0.814	0.975	1.1	1.3	1.5
300	0.009	0.371	0.733	1.1	1.5	1.8	2.2	2.5	2.9	3.3
400	0.011	0.654	1.3	1.9	2.6	3.2	3.9	4.5	5.1	5.8
500	0.014	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
600	0.016	1.5	2.9	4.3	5.8	7.2	8.7	10.1	11.6	13.0
700	0.019	2.0	3.9	5.9	7.9	9.8	11.8	13.8	15.7	17.7
800	0.021	2.6	5.1	7.7	10.3	12.8	15.4	18.0	20.5	23.1
900	0.024	3.3	6.5	9.8	13.0	16.2	19.5	22.7	26.0	29.2
1000	0.026	4.0	8.0	12.0	16.0	20.0	24.1	28.1	32.1	36.1

Table 5.7: Memory needed to store a SIREN neural network with a given layer size (LS) and number of hidden layers (HL) in MBs

hidden layers and a layer size of 250 (which needed 2.8MB space), it should be possible to store much larger road graphs in our model with a high accuracy.

Models with graph embedding techniques

From the literature, many models [20, 19, 32] that utilized graph embedding techniques did not use deep neural networks like we did. To see what influence different embedding dimensions have on the effective used memory, we fixed the number of hidden layers to two.

LS / Embedding Dimension	**8**	**16**	**32**	**64**	**128**
100	0.082	0.121	0.198	0.352	0.66
200	0.21	0.255	0.345	0.524	0.883
300	0.417	0.469	0.571	0.777	1.2
400	0.705	0.763	0.878	1.1	1.6
500	1.1	1.1	1.3	1.5	2.0
600	1.5	1.6	1.7	2.0	2.6
700	2.0	2.1	2.3	2.6	3.2
800	2.7	2.7	2.9	3.2	3.9
900	3.3	3.4	3.6	4.0	4.7
1000	4.1	4.2	4.4	4.9	5.6

Table 5.8: Memory needed to store a model with an embedding layer with fixed two hidden layers and different layer sizes (LS) and embedding dimensions (ED) in MBs

Comparing this table to the previous table, we see that the embedding layer is responsible for a large part of the memory used, but this proportion becomes smaller when we scale up the number of parameters in the model itself. However, looking at the table as a whole, we see that the embedding layer uses a negligible amount of memory.

Chapter 6

Conclusion

6.1 Conclusion

In this thesis, we investigated if a simple multi layer perceptron (MLP) can be trained to memorize the distances in simple graphs. It was able to do so for small graphs but failed when we scaled up the number of nodes. From our tests, we noticed that the number of parameters in the MLP scaled quadratically. This would mean that it is not feasible to use multilayer perceptrons to predict distances in graphs that have more than 100 nodes. However, changing the activation function of our model to a periodic one enabled us to learn finer details while still having a reasonable model size. This model was able to precisely memorize the road graph of Zürich with a very deep but small model. When the graph size exceeds 1000 nodes, then our model already uses less space than a lookup table, which gives hope for future research.

There are a number of models already proposed [20, 19, 32] to predict the distances in a road graph. However, these models all use different embeddings and/or complex model structures, but they also had a different focus. We wanted to see what it takes to predict the distances perfectly, while they tried to build a competitive distance predictor. These models have the benefit that they need less time to train. On the other hand, our model shines on its simple non-specialized structure, leaving the door open for much larger graphs or other applications where it is necessary to memorize very fine details.

6.2 Future work

In this thesis, we were not able to draw any general conclusions as to how exactly simple models need to change, to achieve good accuracy, when we scale up the graph size. An interesting project would be to find out why this is the case. An other project would be to use our model on larger graph datasets such that one has a direct comparison with similar models that were suggested in the literature.

Bibliography

[1] https://sites.google.com/a/pressatgoogle.com/google-maps-for-iphone/google-maps-metrics, 06.07.2021.

[2] https://www.facebook.com/iq/insights-to-go/2740m-facebook-monthly-active-users-were-2740m-as-of-september-30, 06.07.2021.

[3] https://static.digitecgalaxus.ch/Files/1/0/5/6/2/1/4/9/2018-01-08-Geschaftsgang-2017-Digitec-Galaxus.pdf, 06.07.2021.

[4] https://users.diag.uniroma1.it/challenge9/download.shtml, 07.07.2021.

[5] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*, pages 349–360, April 2013.

[6] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems (NIPS)*, pages 585–591, January 2001.

[7] Shiri Chechik. Approximate distance oracles with improved bounds. In *STOC*, June 2015.

[8] Soumi Das, Rajath Nandan Kalava, Kolli Kiran Kumar, Akhil Kandregula, Kalpam Suhaas, Sourangshu Bhattacharya, and Niloy Ganguly. Map enhanced route travel time prediction using deep neural networks. November 2019.

[9] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. April 2021.

[10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, May 2010.

[11] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 855–864, July 2016.

[12] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems Science and Cybernetics*, volume 4, July 1968.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, February 2015.

[14] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). In *ICCV*, June 2016.

[15] Nathan Linial. Finite metric spaces–combinatorics, geometry and algorithms. In *Proc. International Congress of Mathematicians*, page 573–586, April 2003.

[16] Shuai Ma, Kaiyu Feng, Haixun Wang, Jianxin Li, and Jinpeng Huai. Distance landmarks revisited for road graphs. January 2014.

[17] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations Workshops (ICLR)*, January 2013.

[18] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. Fast shortest path distance estimation in large networks. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)*, page 867–876, March 2009.

[19] Jianzhong Qi, Wei Wang, Rui Zhang, and Zhuowei Zhao. A learning based approach to predict shortest-path distances. January 2020.

[20] Fatemeh S. Rizi, Joerg Schloetterer, and Michael Granitzer. Shortest path distance approximation using deep learning techniques. In *IEEE/ACM International Confer-ence on Advances in Social Networks Analysis and Mining (ASONAM)*, page 1007–1014, February 2020.

[21] Chaiyaphum Siripanpornchana, Sooksan Panichpapiboon, and Pimwadee Chaovalit. Travel-time prediction with deep learning. In *2016 IEEE Region 10 Conference (TENCON)*, November 2016.

[22] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems 33*, June 2020.

[23] Gayatri Swamynathan, Christo Wilson, Bryce Boe, Kevin Almeroth, and Ben Y. Zhao. Do social networks improve e-commerce?: a study on social marketplaces. In *Proceedings of the first workshop on Online social networks. ACM*, pages 1–6, August 2008.

[24] Frank W. Takes and Walter A. Kosters. Adaptive landmark selection strategies for fast shortest path computation in large real-world graphs. In *Proceedings of the IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies (WI-IAT)*, page 27–34, August 2014.

[25] Qiang Cui Yuyuan Tang, Shu Wu, and Liang Wang. Distance2pre: Personalized spatial preference for next point-of-interest prediction. In *Yang Q., Zhou ZH., Gong Z., Zhang ML., Huang SJ. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2019*, March 2019.

[26] Mikkel Thorup and Uri Zwick. Approximate distance oracles. In *J. ACM 52*, pages 1–24, February 2005.

[27] Monique V. Vieira, Bruno M. Fonseca, Rodrigo Damazio, Paulo B. Golgher, Davi de Castro Reis, and Berthier Ribeiro-Neto. Efficient search ranking in social networks. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management. ACM*, pages 563–572, November 2007.

[28] Daixin Wang, Peng Cui, , and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conferenc on Knowledge Discovery and Data Mining (SIGKDD)*, pages 1225–1234, August 2016.

[29] Dong Wang, Junbo Zhang, Wei Cao, Jian Li, and Yu Zheng. When will you arrive? estimating travel time based on deep neural networks. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, April 2018.

[30] Mengjia Xu. Understanding graph embedding methods and their applications. December 2020.

[31] Yuan Yuan, Chunfu Shao, Zhichao Cao, Zhaocheng He, Changsheng Zhu, Yimin Wang, and Vlon Jang. Bus dynamic travel time prediction: Using a deep feature extraction framework based on rnn and dnn. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, November 2020.

[32] Zhuowei Zhao. Embedding graphs for shortest-path distance predictions. In *30th Symposium on Comparative Irrelevance, Somewhere, Some Country*, February 2020.

[33] Tianyu Zhou. Deep learning models for route planning in road networks. November 2019.

Appendix A

Plots from chances and limitations of simple graphs

A.1 Claw Graph

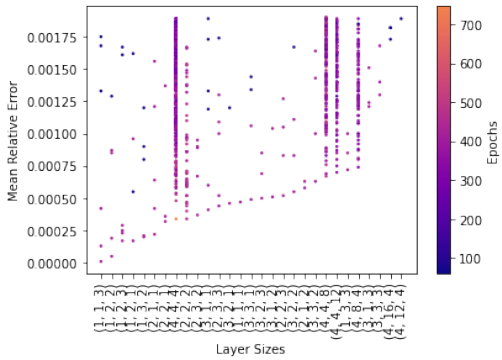

Figure A.1: Comparison of layer sizes

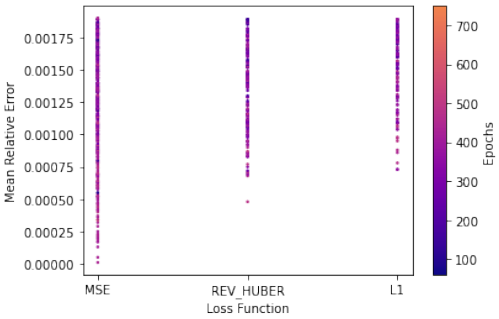

Figure A.2: Comparison of Loss Functions

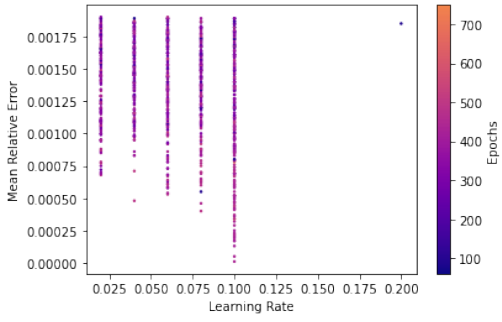

Figure A.3: Comparison of learning rates

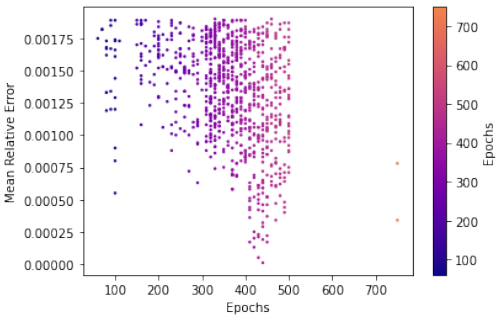

Figure A.4: Comparison of epochs

A.2 10x10 Grid Graph

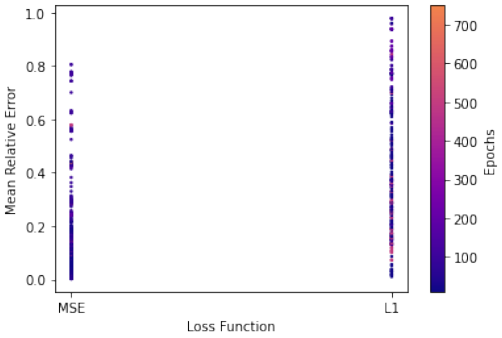

Figure A.5: Comparison of loss functions (outliers excluded)

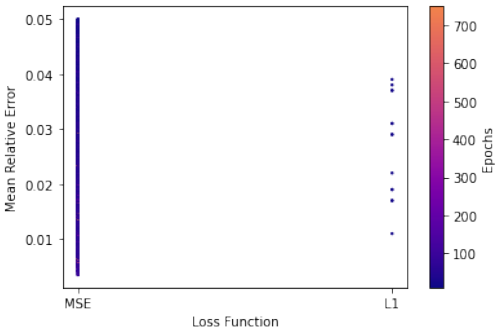

Figure A.6: Comparison of loss functions (only best results)

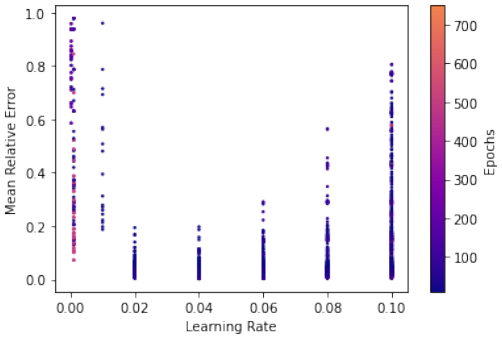

Figure A.7: Comparison of learning rates (outliers excluded)

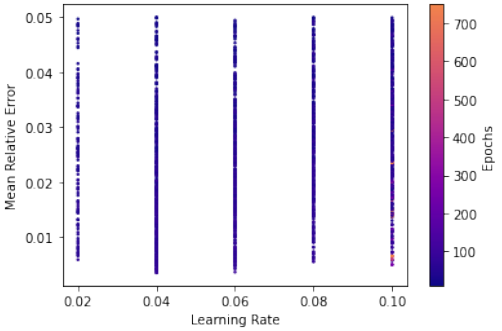

Figure A.8: Comparison of learning rates (only best results)

A.3 Tree

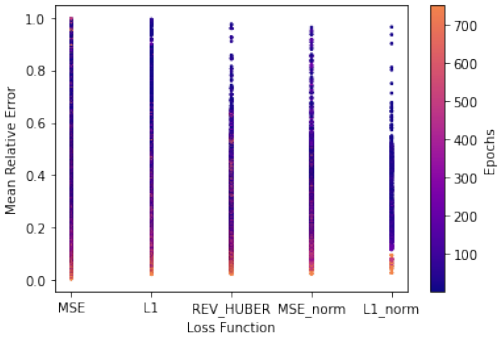

Figure A.9: Comparison of loss functions (outliers excluded)

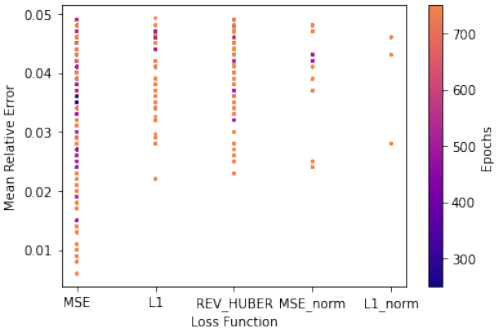

Figure A.10: Comparison of loss functions (only best results)

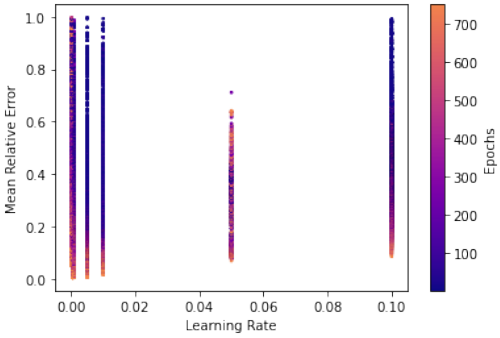

Figure A.11: Comparison of learning rates (outliers excluded)

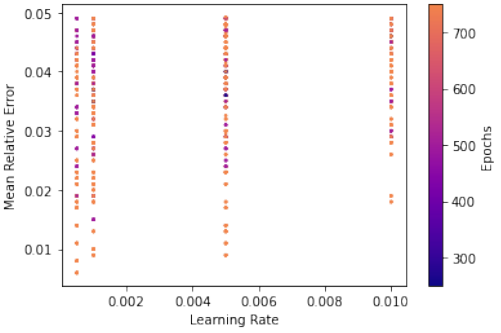

Figure A.12: Comparison of learning rates (only best results)

A.4 Random geometric graph

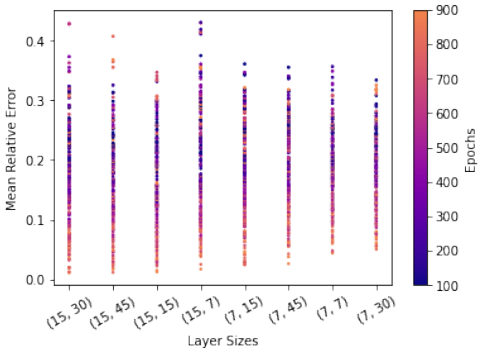

Figure A.13: 7 nodes: Comparison of layer sizes (of best 50% of runs)

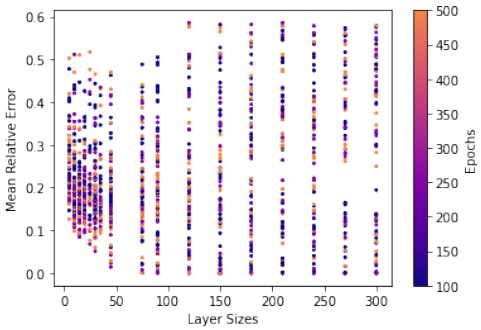

Figure A.14: 15 nodes: Comparison of Learning Rates (of best 15% of runs)

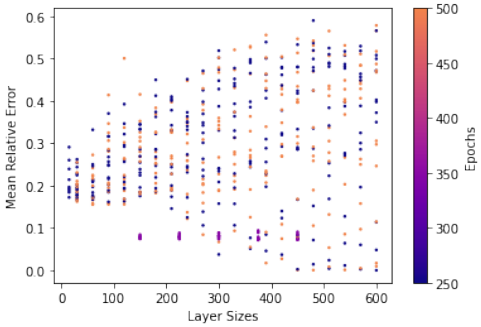

Figure A.15: 30 node: Comparison of layer sizes (of best 25% of runs

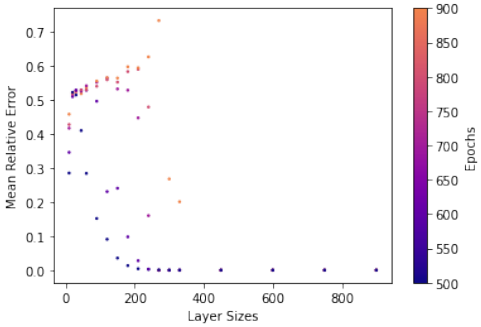

Figure A.16: 45 node: Comparison of Learning Rates (of best 50% of runs)

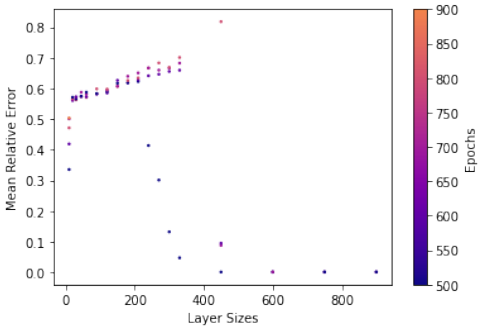

Figure A.17: 60 node: Comparison of layer sizes (of best 50% of runs

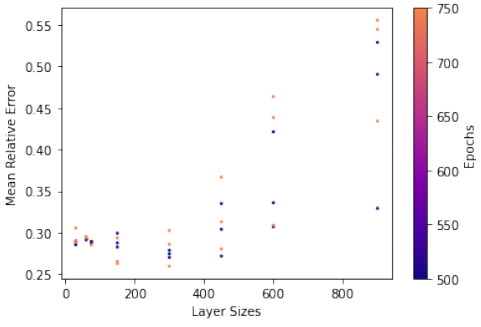

Figure A.18: 75 node: Comparison of Learning Rates (of all runs, without extreme outliers)

Appendix B

Plots from SIREN model

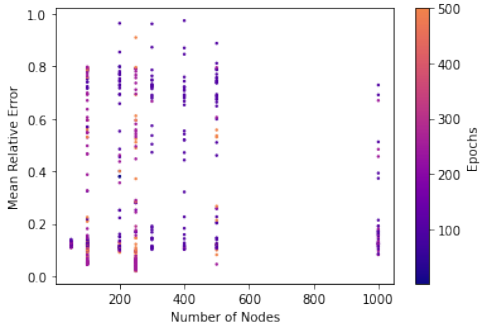

Figure B.1: SIREN: Layer sizes of all tests

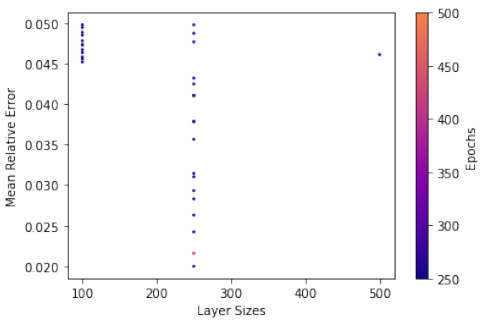

Figure B.2: SIREN: Layer sizes of good tests

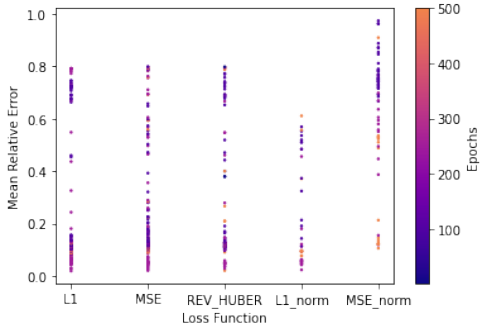

Figure B.3: SIREN: Loss functions of all tests

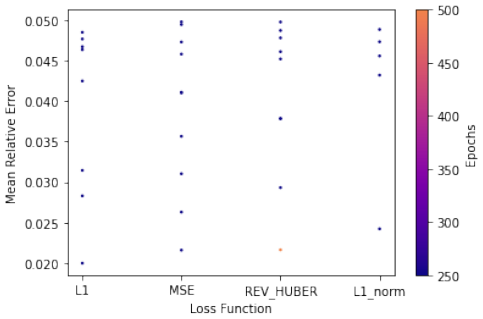

Figure B.4: SIREN: Loss functions of good tests