Graph-based Sparse Neural Networks for Traffic Signal Optimization

Lukasz Skowronek[2], Pawel Gora[1,2], Marcin Mozejko[2] and Arkadiusz Klemenko[2]

[1]*Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland*
[2]*TensorCell*

Abstract

We investigate the performance of sparsely connected neural networks, with connectivity determined by road network graphs, for solving the Traffic Signal Setting optimization problem. We conducted experiments on three realistic road network topologies and found these types of graph neural networks superior to fully connected ones, both in terms of generalization properties on fixed test sets and - more importantly - near target function minima obtained in the gradient descent optimization process. We additionally confirm the soundness of our method by showing that random perturbations of the actual graph lead to consistent deterioration of model performance.

Keywords

traffic optimization, graph neural networks, Traffic Signal Setting problem, surrogate modelling

1. Introduction

Traffic optimization problems have a natural underlying graph structure, determined by the topology of the corresponding road network. In this paper, we introduce a neural network architecture based on a road network graph adjacency matrix to solve the so-called Traffic Signal Setting (TSS) problem, in which the goal is to find the optimal traffic signal settings for given traffic conditions (as defined in [1]). Some variants of this problem were proven to be NP-hard even for very simple traffic models ([2]), and therefore, heuristics and approximations have been used to solve it ([1]), but the existing approaches still have some drawbacks. For example, evaluating the quality of traffic signal settings using accurate traffic simulations (which is a standard evaluation method) can be too time-consuming, especially for large-scale road networks and/or online evaluation ([3, 4]). Also, the size of the space of possible solutions is so large that it turns out infeasible, in any reasonable time, to obtain global minima (or even a relatively good signal settings) of the simulator output by checking all the possible solutions or doing a random search ([1]), as most points in the input space are far from the optimal solutions.

A strategy used for solving these two difficulties was presented in [1] and consists of generating a reasonably sized training set using a traffic simulator and then fitting a machine learning

29th International Workshop on Concurrency, Specification and Programming (CS&P'21)
p.gora@mimuw.edu.pl (P. Gora)
https://www.mimuw.edu.pl/~pawelg (P. Gora)
0000-0002-8037-5704 (P. Gora)

model to approximate the outcomes of traffic simulations very fast and accurately. The output of these models can be then minimized using an optimization algorithm, such as gradient descent, in hope to obtain close to optimal traffic signal settings. This strategy turned out to be quite successful ([1, 5, 4]), yet the models' accuracy degraded close to points considered as minima by the optimization algorithm and the model, making further optimization far more difficult ([3, 4]).

In this paper, we show that the graph-based neural networks (GNN) that we use, built based on road network graphs, can outperform feed-forward fully connected neural networks (FCNN) on this task. Comparing to the standard FCNNs, the introduced GNNs have most of the connections deleted, keeping only the crucial connections between neurons (making the information flow corresponding to the traffic flow in the road network), which makes this architecture relatively sparse, easier to train and generalize. As a consequence, GNNs have better accuracy on the test set, as well as close to the local optima found using gradient descent optimization applied to the TSS problem. We also prove that GNN architectures work better than analogous ones built based on perturbed adjacency matrices.

The rest of the paper is organized as follows. Section 2 puts our work in the context of a research in the domain of building surrogate models for complex processes, solving TSS problem and using graph neural networks. Section 3 presents the two types of graph neural network architectures we used. In Section 4, we describe the setup of our main experiments including a description of the used datasets and their generation. Section 5 summarizes our main experiment results showing that neural network architectures introduced in this paper outperform other models used in such a task. In Section 6, we summarize the results of several 'sanity checks' we performed in order to confirm that our results were not obtained by chance. Moreover, the results in Section 6 can be especially interesting for researchers in graph neural networks, e.g. we show that out-of-sample performance of graph-based sparse neural nets decreases (almost) monotonously as a function of the distance of the adjacency matrix we use for constructing the network from the true adjacency matrix. We summarize the presented research in Section 7, outlining some possible future research directions.

2. Related works

Complex processes, such as road traffic in cities, are difficult to study due to large number of interacting components (e.g., vehicles), nondeterminism or sensitive dependence on initial conditions. Very often, the only reasonable method to accurately predict the behaviour of such systems is to apply computer simulations which can be time-consuming and usually can't be simplified due to computational irreducibility. However, in many tasks related to complex processes it is not necessary to obtain very accurate predictions, it can be sufficient to get approximate outcomes but as fast as possible (due to stochasticity or sensitive dependence on initial conditions, it may be impossible to predict the exact value anyway). Therefore, in such cases it is natural to build the so-called surrogate models (metamodels) approximating outcomes of simulations very fast and with a good accuracy ([6]). Such applications are especially common in the case of optimization tasks, in which quite often it is necessary to run multiple simulations in order to evaluate many different input settings ([7, 8, 9]). This is the case of traffic optimization

problems [10], such as the TSS problem. Many such surrogate models are based on machine learning methods, such as neural networks [7, 8], and also some previous works on solving TSS [1, 5, 3, 4] use various machine learning techniques (e.g., based on neural networks or gradient boosted decision trees) to build metamodels of traffic simulations, which were used to evaluate quality of traffic signal settings. Such metamodels were able to approximate outputs of simulations (the total times of waiting on red signals) with a very good accuracy (e.g., values of the MAPE metrics were at the level $1 - 2\%$) and a few orders of magnitude faster than by running microscopic simulations [1, 5]. Thanks to that, it was possible to use optimization algorithms such as genetic algorithms or gradient descent, to find heuristically optimal signal settings without performing extensive parameter space searches that would take weeks to complete [1, 5, 3, 4]. However, information about the road network structure has never been used in those experiments, even though it should naturally be relevant when optimizing traffic.

Introducing the direct connection between the network architecture and the graph structure can help to leverage additional information represented by a graph. Similarly to our work, [11] introduces a graph NN layer in which each vertex has specific parameters assigned to combine information from its neighbors. However, differently to our method, this layer uses only an original graph matrix and skips the dual graph structure when performing computations. The notable usage of a dual structure can be found in [12], where it is compressed to a PPMI matrix using aggregated statistics from a random graph walk procedure. This aggregation is used to introduce a vertex neighborhood context similarly to a popular T-SNE method [13]. [14] provides an extensive overview of different graph neural networks architectures and applications.

Due to their capability to capture a road-network structure, GNNs were used in multiple traffic applications. In [15, 16, 17] authors used spatio-temporal GNNs for a traffic situation prediction, whereas [18] used the same technique in order to predict the TAXI demand. However, our application of graph neural networks in the traffic optimization domain and the Traffic Signal Setting problem seems to be the first such approach.

3. Network architecture

The key idea in defining our sparse graph-based neural network architecture is an intuitively compelling rule that information/signal should propagate locally between the net layers. By locality, we mean the presence of only those neuron connections that have a corresponding non-zero entry in the adjacency matrix of the corresponding graph. In the case of the road network, in order to implement such a rule, the neurons in the successive layers of the neural network should be linked to the neurons corresponding to vertices and/or edges of the corresponding graph. Thus, we propose the following general ways to build a graph neural network (see Section 1 of Supplementary materials ([19]) for mathematical formulas):

1. Neurons in the even numbered layers, starting with the input layer as layer 0, correspond to graph vertices (in our case - road crossings). Neurons in the odd numbered layers correspond to graph edges (in our case - road segments). An exception should be the final layer with just one neuron. Connections from a vertex-localized layer to an edge-localized layer should only be present if a given vertex is an end of a given edge in the corresponding road network graph. There are exactly two such connections for every edge

neuron. Connections from an edge-localized layer to a vertex-localized layer should only be present if the edge has the vertex as its end in the corresponding road network graph. The number of such connections is equal to the number of particular vertex neighbors.

or

2. Neurons in all layers, with the exception of the output layer, correspond to road network graph vertices. Connections from a neuron in one layer to a neuron in the next one should only be present if the corresponding vertices are neighbors in the road network graph. The number of connections for the vertex node is equal to the number of the vertex neighbors.

Although architecture 2 might seem to be more basic, architecture 1 appears to naturally model a traffic flow through the road networks (see Supplementary materials ([19]), Section 2, for a detailed explanation). In the rest of this paper, we focus solely on GNNs of the architecture type 1.

It should also be pointed out that GNNs can have multiple channels at each edge/vertex. The number of channels in each layer is a hyperparameter of the network. In the following part, we always assume the number of channels to be constant across the hidden layers of the network.

One may also notice a similarity between our GNN architecture 2 and the graph neural networks proposed by Thomas Kipf [20]. However, we do not share any weights in our model, as we aim to focus on local patterns connected to roads / crossroads. Theoretically, we could introduce some weight sharing in the 'edge' layers of GNNs of type 1, but our first experiments using this approach led to highly disappointing results.

In typical ML literature terminology, our GNNs should likely be called 'NNs with a fixed sparse connectivity mask'. In case of multi-channel networks, sparsity is applied in the 'spacial', but not in the 'channel' dimension.

4. Experiment setup

In order to train the surrogate models, it was necessary to generate datasets first. For this task, we simulated vehicular traffic on 3 realistic road networks, corresponding to selected districts in Warsaw: Centrum, Ochota and Mokotów, including 11, 21 and 42 intersections with traffic signals, respectively. The simulations were run using a microscopic traffic simulator, Traffic Simulation Framework [21], for which a road network description for Warsaw was obtained from the OpenStreetMap service [22]. The inputs to the simulator were vectors of lengths 11, 21 and 42, respectively. Each position in a vector represented an offset of a traffic signal on a corresponding intersection. The offsets are shifts with respect to a global two minute traffic signal cycle start - times from the beginning of the simulation to the first switch from the green signal state to the red signal state (it was assumed for simplicity that the duration of a green signal phase is always equal to 58 seconds, while duration of a red signal phase is equal to 62, constituting a 120-second cycle). The offsets were provided as integers, measured in seconds, hence they ranged from 0 to 119 (note the periodicity of these variables). The simulator output in each case was the total waiting time on red signals, summed for all the cars participating in the simulation on a considered area (finding the inputs minimizing this output value was the optimization goal of the considered TSS problem instance).

Each simulation lasted 10 minutes and consisted of 42000 cars on the whole road network of Warsaw. The datasets for Ochota, Mokotów and Centrum were generated using approximately 100000 randomly selected inputs for the TSF simulator (the input offset values from the set $\{0, 1, \ldots, 119\}$ were sampled from the uniform distribution independently). These datasets are publicly available to enable further research ([23]).

After preparing the datasets, we trained GNN and FCNN networks as metamodels to solve TSS using gradient descent. Before training, we scaled the inputs to $[-1, 1]$ using the following mappring $x \mapsto \sin(2\pi x/120)$ and $x \mapsto \cos(2\pi x/120)$, thus doubling the input size (actually, increasing the number of input channels). This is motivated by the periodicity of the problem - the neural network may learn that the offsets are periodic and values 0 and 120 correspond to the same setting and this can improve the training [3, 4]. For the output, we used a standard scaler that divides the data by its standard deviation and shifts the mean to zero.

For each of the 3 considered road networks, we tested 9 different GNN architectures and 9 FCNN architectures. The 9 selected GNN architectures corresponded to all combinations of values from the following parameter sets:

- number of hidden GNN layers: 2, 3, 4 (not counting input and output layers);
- number of channels per layer: 3, 4, 5.

The activation function we used was `tanh`, indicated as superior to ReLU in preliminary experiments and in previous works [4].

For comparison, we also tested 9 FCNN architectures with `tanh` activation function, covering all combinations of parameter values from the following sets:

- number of hidden layers: 2, 3, 4
- number of neurons per layer: 20, 40, 100

For each of the 3 datasets, we used the same 90/10 train/test split for each of the considered 18 hyperparameter settings (9 GNN architectures and 9 FCNN architecture). For each of the architectures, we ran the following procedure:

1. Train a model on the training set for about 1100 epochs (concretely, minimize on 10^5 random mini-batches of size 997 (997 is the closest prime to 1000 - a prime number was chosen to assure better randomization, although it was not expected to have any real effect) using Adam optimizer ([24]) and a learning rate of 0.0035).
2. Evaluate the trained model on the test set using the mean relative error with respect to the original outputs as the core metrics.
3. Generate 100 gradient descent trajectories of the trained model output with respect to its inputs (in the original input space, backpropagating through the sin/cos transformation). Gradients were evaluated at inputs rounded in the original parameter space (our traffic simulator (TSF) accepts only integer inputs). Nesterov updater ([25]) with a learning rate of 0.01 and momentum of 0.9 is used. Each trajectory had 3000 steps. This is similar to approach used in [4].
4. Every 30 steps, transform the current trajectory point to the original parameter space, round and send to the TSF simulator. Save the inputs and the simulator outputs to a new 'simulation' test set.

5. Evaluate the trained model on the 'simulation' test set using various metrics (cf. the discussion in Section 5).

All the experiments were run on virtual machines in the Azure cloud (NC6 with NVIDIA Tesla K80 ([26])). The code used in the experiments can be found at ([27]). All of the models trained for the main experiment and all the out-of-sample simulation datasets can be found at [28]. The core dataset can be obtained at [23].

5. Experiment results

Table 1
Core results for the three best GNN and the three best FCNN architectures according to the accuracy (MAPE) on the test set (i.e. gradient descent results did not affect the selection of these models).

Measure	Model	Ochota	Mokotów	Centrum
Min. MAPE	GNN	1.33%	0.76%	0.80%
on the test set	FCNN	1.71%	0.94%	0.87%
Min. simulation output	GNN	32,205	265,129	63,606
	FCNN	32,587	266,237	63,553
Avg MAPE on the lowest	GNN	1.26%	0.53%	0.76%
5% sim. outputs	FCNN	5.35%	3.04%	2.49%
Avg MAPE on the lowest	GNN	1.75%	0.84%	1.22%
10% sim. outputs	FCNN	4.53%	2.74%	2.25%
Avg MAPE on the lowest	GNN	1.51%	1.00%	1.11%
15% sim. outputs	FCNN	4.65%	2.56%	2.04%

The key results of our experiments with GNNs are shown in Table 1, as well as in Figure 1, complemented by the tables and figures in Section 3 of Supplementary Materials [19]. Table 1 shows a summary of core performance measures, calculated for three top GNN and three FCNN, ranked based on the average accuracy on the test set (MAPE). The core measures presented are:

- Minimum MAPE (mean absolute percentage error) on the test set. This number can be obtained *before* doing gradient descent. The minimum is taken among the 3 top ranked GNNs or FCNNs (according to the row description). Because of the model selection criterion we use for Table 1, this minimum is global within the respective 9-element model universe (GNN or FCNN).
- Minimum simulation output obtained when doing gradient descent (note that while being interesting from a traffic optimization perspective, this measure lacks robustness, as it can be distorted by a single data point).
- Average MAPE on $x\%$ (for $x = 5, 10, 15$) gradient descent trajectory ends, selected according to the corresponding simulator output value (sorted lowest first). An average is taken over the three models selected, GNN or FCNN, according to the row description.

First, let us note that the results of Table 1 show a better performance of GNNs comparing to FCNNs, particularly in terms of minimum MAPE of the test set and average MAPE on the lowest

points from the gradient descent trajectory according to the simulation. The improvement is visible for all the 3 road maps (Ochota, Mokotów, Centrum) and all the 5 core measures (with the exception of the minimum simulation output value obtained for Centrum, where one FCNN turned out to yield slightly (less than 0.1%) lower result than all the GNNs).

To summarize, the core improvement areas are:

- Much lower error on the test set.
- Lower minimum simulator output value obtained when doing the gradient descent (except for Centrum, for which we can count a draw).
- Much lower approximation error obtained on the trajectory ends corresponding to 5%, 10% and 15% lowest simulator output values.

Figure 1 as well as similar figures for Ochota and Mokotów (see Section 3 of Supplementary Materials [19]) show the density of gradient descent trajectory points as heatmap plots. The horizontal axis corresponds to gradient descent trajectory point number (recorded every 30 steps), and the vertical axis corresponds to the simulator output. Each trajectory had 3000 steps, but we recorded points every 30 steps. The plots show a heatmap of these points on the (point number, simulator output) plane. Thus, the more points in some area, the brighter the color. Also, if one architecture reaches a lower minimum than another, the resulting heatmap is taller.

Besides confirming some of the quantitative conclusions from Table 1, the heatmaps also show that in many cases, the gradient descent is less 'noisy' for GNN, suggesting a smoother function surface, less prone to overfit noise (this is best visible in the plots in Section 3 of [19]).

6. Consistency checks

The findings of the previous section call for some careful consistency checks before reaching final conclusions. In particular, it is not fully clear that the actual adjacency matrix gives any value. Perhaps, any similar graph, even not related to the problem at hand, can do equally well.

To address that question, we decided to fix the number of layers to 3 and the number of channels to 4 per layer (for GNN of type 1), and built our nets using random graphs with various degrees of resemblance to the true problem graph (we repeated this for all the three road networks we considered). As a measure of graph similarity, we used the symmetric difference between the sets of graph edges. The random graphs were generated in two ways. The first method (referred later as **'Edge/Non-edge switching'**) used random edge insertions and deletions, with the desired value of the symmetric difference kept fixed. The second method (referred later as **'Vertex label permutation'**) used random permutations of the vertex labels while keeping the connection graph structure exactly the same. Graphs generated by this method were isomorphic, but not identical to the original one.

It is worth mentioning that although the first method generates truly random graphs similar to the original one, the new graph might not represent a plausible road network. The second method, on the contrary, always keeps the same, realistic road network graph structure, but it provides spurious insights to the training algorithm as crossroads are switched.

Results obtained by the two methods are shown in Figure 2, including Ochota, Mokotów and Centrum. The plots show that the mean relative error achieved on the test sets by neural nets

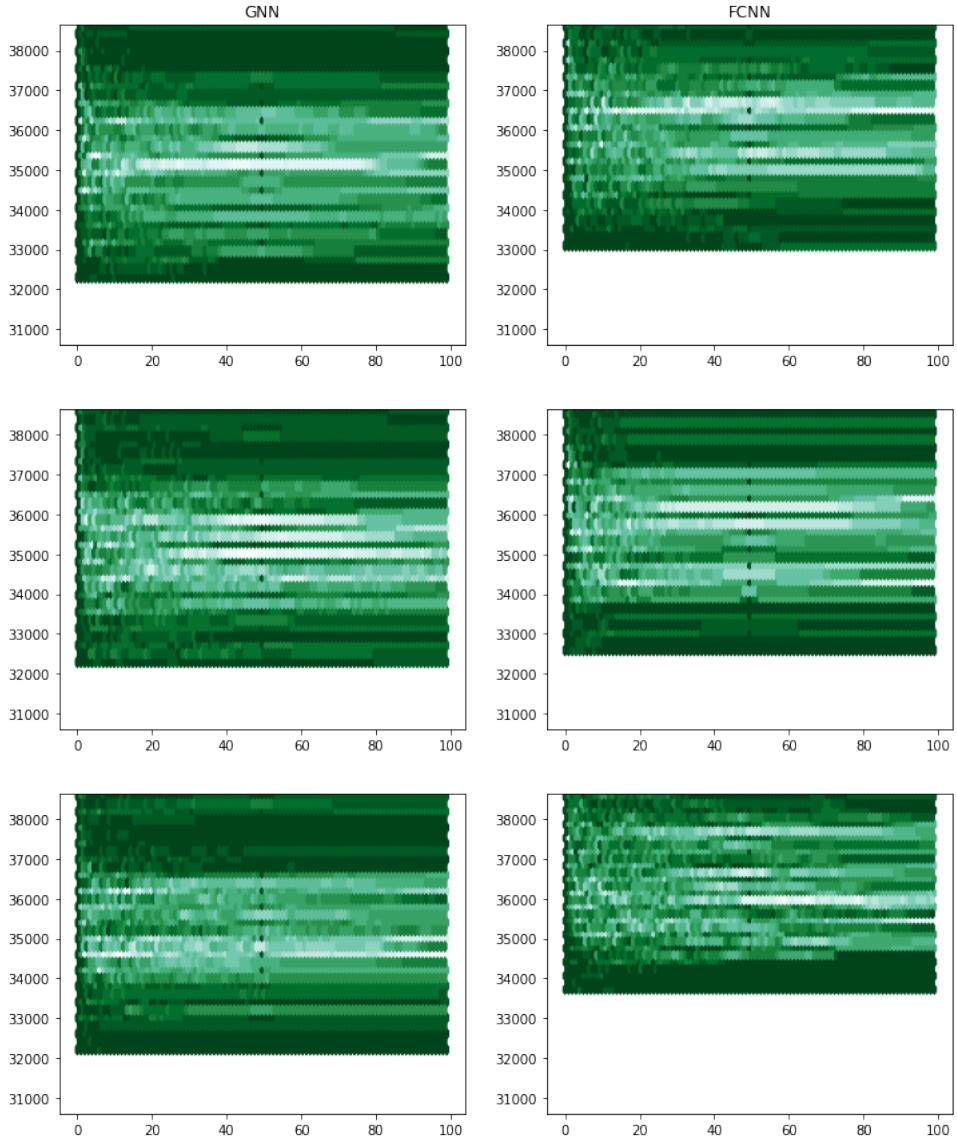
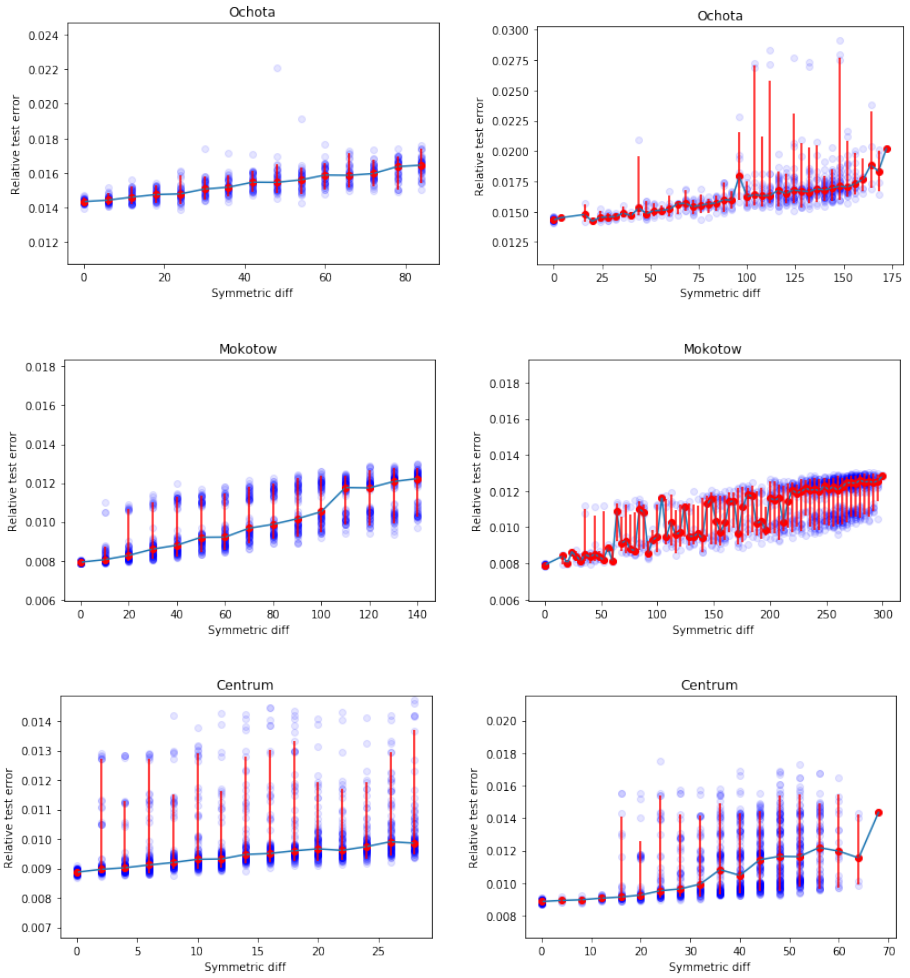


Figure 1: Gradient descent trajectory density plots for Warsaw Ochota for the 3 best GNN and FCNN models. Horizontal axis corresponds to trajectory point number (recorded every 30 steps), vertical axis to simulator output value.

based on random graphs, after roughly 330 epochs of training, as a function of the distance of the graph used for constructing the net to the true graph. The distance was measured using the symmetric difference between the respective edge sets.

As we can see, the median of the mean relative error, denoted with a red dot, grows almost monotonously as a function of the distance of the graph we use to the actual problem graph. This is visible for both graph sampling methods. The minimum average relative error attained for a particular value of the distance also grows, perhaps with a bit more of noise.



(a) Edge/Non-edge switching

(b) Vertex label permutation

Figure 2: Mean relative error achieved by a GNN on test set after roughly 330 epochs of training, shown as a function of the distance of a random graph to the true one. In subfigure 2(a), edge/non-edge switching (described in the text) was used for generating random graphs. In subfigure 2(b), vertex label permutation was used. Red dots denote median result. Errorbars correspond to 5% quantiles.

7. Conclusions

We demonstrated the usefulness of sparsely connected neural networks, with sparsity based on an adjacency graph, in a problem from the traffic optimization domain. GNN consistently outperformed FCNN on fixed test sets for the three realistic road networks we considered (Ochota, Mokotów and Centrum districts in Warsaw). More importantly, GNN achieved approximation quality far superior to FCNN near unseen simulator output value minima. By using randomly perturbed graphs, we also showed that the choice of the proper graph when constructing a GNN is important for achieving good results on a test set.

The kind of NN sparsity considered in this paper, where only some of the connections are allowed, may be regarded as a kind of a regularizer based on the problem graph. It is similar to L1 regularization of a fully connected neural network in that it keeps only some weights non-zero in the trained model. The resulting networks have much fewer parameters than analogous fully connected networks and turn out to generalize significantly better than any architecture we considered so far for solving the TSS problem.

Acknowledgments

The presented research was supported by Microsoft's "AI for Earth" computational grant.

References

[1] P. Gora, K. Kurach, Approximating traffic simulation using neural networks and its application in traffic optimization, in: NIPS 2016 Workshop on Nonconvex Optimization for Machine Learning: Theory and Practice, 2016.

[2] C. Yang, Y. Yeh, The model and properties of the traffic light problem, in: Proc. of International Conference on Algorithms, 1996, pp. 19–26.

[3] P. Gora, M. Brzeski, M. Możejko, A. Klemenko, A. Kochański, Investigating performance of neural networks and gradient boosting models approximating microscopic traffic simulations in traffic optimization tasks, in: "NeurIPS 2018 Workshop "Machine Learning for Intelligent Transportation Systems", 2018.

[4] M. Możejko, M. Brzeski, L. Madry, L. Skowronek, P. Gora, Traffic signal settings optimization using gradient descent, Schedae Informaticae 27 (2018).

[5] P. Gora, M. Bardoński, Training neural networks to approximate traffic simulation outcomes, in: 2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), IEEE, 2017, pp. 889––894.

[6] J. Zhang, S. Chowdhury, J. Zhang, A. Messac, L. Castillo, Adaptive hybrid surrogate modeling for complex systems, AIAA J (2013) 643–656.

[7] D. Rijnen, J. Rhuggenaath, P. R. d. O. d. Costa, Y. Zhang, Machine learning based simulation optimisation for trailer management, in: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), 2019, pp. 3687–3692. doi:10.1109/SMC.2019.8914329.

[8] R. D. Hurrion, A sequential method for the development of visual interactive meta-simulation models using neural networks, The Journal of the Operational Research Society 51 (2000) 712–719.

[9] R. R. Barton, M. Meckesheimer, Chapter 18 metamodel-based simulation optimization, in: S. G. Henderson, B. L. Nelson (Eds.), Simulation, volume 13 of *Handbooks in Operations Research and Management Science*, Elsevier, 2006, pp. 535 – 574.

[10] C. Osorio, M. Bierlaire, A surrogate model for traffic optimization of congested networks: an analytic queueing network approach, in: EPFL-REPORT-152480, 2009.

[11] A. Micheli, Neural network for graphs: A contextual constructive approach, IEEE Transactions on Neural Networks 20 (2009) 498–511.

[12] C. Zhuang, Q. Ma, Dual graph convolutional networks for graph-based semi-supervised

classification, in: Proceedings of the 2018 World Wide Web Conference, WWW '18, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2018, p. 499–508. URL: https://doi.org/10.1145/3178876.3186116. doi:10.1145/3178876.3186116.

[13] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, Journal of Machine Learning Research 9 (2008) 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[14] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu, A comprehensive survey on graph neural networks, CoRR abs/1901.00596 (2019). URL: http://arxiv.org/abs/1901.00596. arXiv:1901.00596.

[15] Y. Li, R. Yu, C. Shahabi, Y. Liu, Graph convolutional recurrent neural network: Data-driven traffic forecasting, CoRR abs/1707.01926 (2017). URL: http://arxiv.org/abs/1707.01926. arXiv:1707.01926.

[16] B. Yu, H. Yin, Z. Zhu, Spatio-temporal graph convolutional neural network: A deep learning framework for traffic forecasting, CoRR abs/1709.04875 (2017). URL: http://arxiv.org/abs/1709.04875. arXiv:1709.04875.

[17] S. Guo, Y. Lin, N. Feng, C. Song, H. Wan, Attention based spatial-temporal graph convolutional networks for traffic flow forecasting, in: AAAI, 2019.

[18] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, J. Ye, Z. Li, Deep multi-view spatial-temporal network for taxi demand prediction, CoRR abs/1802.08714 (2018). URL: http://arxiv.org/abs/1802.08714. arXiv:1802.08714.

[19] Supplementary, Supplementary materials, 2021. URL: https://drive.google.com/file/d/1sba_cunGhao4z4-loIfQYV7u4cXCdnWk/view?usp=sharing.

[20] T. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017, Conference Track Proceedings, 2017.

[21] P. Gora, Traffic simulation framework - a cellular automaton based tool for simulating and investigating real city traffic, in: Recent Advances in Intelligent Information Systems, 2009, pp. 641–653.

[22] OSM, Openstreetmap, 2021. URL: https://www.openstreetmap.org.

[23] Dataset, Dataset used for experiments, 2021. URL: https://drive.google.com/file/d/1aLUL3QPxGxeUVmqds6HWeGnVnQ5O4Mxr/view?usp=sharing.

[24] D. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Proceedings of the 3rd International Conference on Learning Representations (ICLR), 2015.

[25] Y. Nesterov, A method for unconstrained convex minimization problem with the rate of convergence o $(1/k^2)$, Doklady AN USSR 269 (1983) 543—-547.

[26] VMs, Description of virtual machines used in experiments, 2021. URL: https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-gpu.

[27] Code, Zipped repository of the code used in our experiments, 2021. URL: https://drive.google.com/file/d/1FF6q8GTJljkYjKSNMYsL5neoXbOqPIcm/view?usp=sharing.

[28] Models, Models trained for the main experiment and all the out-of-sample simulation datasets, 2021. URL: https://drive.google.com/file/d/1mPnFt1Y1wGLGE-ha2_JiYsuJEebnfBYx/view?usp=sharing.