Spatio-Temporal Forecasting with Time-Adaptive Graph Neural Networks

Varun Ursekar
Carnegie Mellon University
vursekar@andrew.cmu.edu

Shalin Shah
Carnegie Mellon University
shalins2@andrew.cmu.edu

1 Introduction

Spatio-temporal forecasting is an important problem that has applications in a variety of fields, including climatology [5], urban systems [3], transportation [12], and business [16]. As a predictive task, it is a special case of multivariate time-series forecasting where the component time-series are assumed to be correlated in accordance with some spatial structure. In a classical traffic forecasting problem, for example, the speed of traffic at nearby road intersections might reasonably be assumed to be correlated with each other.

The key challenge that spatio-temporal forecasting poses is how to integrate information from an individual signal's past with the information provided by other signals. Since these temporal and spatial relationships can often be non-linear [2], basic linear models often fail to achieve good performance. Traditional time-series models such as ARIMA and VAR, for example, might have to be combined with kernel methods to model certain problems. As in other fields, this has motivated the use of deep learning techniques that can learn these possibly non-linear relationships directly from the data.

While traditional deep learning methods for sequential data such as RNNs and temporal convolutional networks integrate temporal information well [1], the integration of spatial information presents additional challenges. Two issues are scale and flexibility: while in theory we can simply "flatten" a matrix-valued signal, this approach would not scale well when dealing with many spatial locations or when the number of nodes might vary at inference time. This has motivated the use of weight-sharing. Past approaches have tried using traditional convolutional kernels to integrate features from proximal nodes [17; 1]. More recent methods turned to more flexible graph convolutions that explicitly take into account the spatial structure in the data, i.e. using graph neural networks [19; 12; 3; 18; 20; 9; 15].

Graph neural networks provide a way to integrate domain knowledge about the underlying spatial structure into our models by way of a pre-specified adjacency matrix $\mathbf{A}$ [10]. In traffic forecasting, one choice for $\mathbf{A}$ might be an $N \times N$ matrix specifying the distances between each intersection. This approach has been shown to be effective when the underlying graph is static and known [12; 19]. However, in many instances, the underlying graph may be difficult to define, and possibly dynamic. In these cases, a pre-defined spatial prior might not capture all spatial dependencies in the data, possibly degrading model performance [3].

One example, which we study here, is that of bike-sharing supply and demand forecasting. Bike-sharing systems have grown significantly in popularity in recent years [4; 13]. Being able to predict where and when people rent bikes is important for optimally distributing resources across the network [6]. Unlike in the traffic forecasting problem, while bike-sharing stations might be close to each other, bikes generally have many ways to go from one station to another. Thus, a graph based on spatial structure alone is unlikely to capture all the dependencies in the data. Previous work has shown that graphs based on spatial proximity, station-to-station trip counts, and correlated demand lead to better performance than graphs based on any one factor alone [4]. This begs the question: is it possible to learn dependencies directly from the data?

10707 Class Project (2022).

In this work, we seek to address the difficulty of specifying pre-defined adjacency matrices by building off of graph neural network architectures such as GraphWaveNet and FC-GAGA, that use *self-adaptive matrices*, i.e. adjacency matrices that are learnable [3; 18; 15]. While previous works allow adjacency matrices to be learned, they do not allow them to change as a function of time; we propose to use time-adaptive matrices that utilize time information to modulate node embeddings, and thereby, the adjacency matrix.

In mathematical terms, our problem is as follows: we are given a matrix-valued signal $\mathbf{X}_t \in \mathbb{R}^{N \times d}$, where N is the number of spatial locations or nodes (e.g. bike-share stations) and d is the dimension of the signal from each node (here 2, i.e. supply and demand of bikes at each station). We would like to find a mapping:

$$f : [\tilde{\mathbf{X}}_{t-S}, \tilde{\mathbf{X}}_{t-S+1}, ..., \tilde{\mathbf{X}}_t] \rightarrow [\mathbf{X}_{t+1}, \mathbf{X}_{t+2}, ..., \mathbf{X}_{t+H}],$$

where S is the number of past signals used to predict the next H signals in the future at time t. Here, either $\tilde{\mathbf{X}}_t = \mathbf{X}_t$ or $\tilde{\mathbf{X}}_t = \mathbf{X}_t || \mathbf{Z}_t$, where $\mathbf{Z}_t$ is a set of covariates that are row-wise concatenated with the signal. The covariates are distinguished from the signal in that they are not predicted by the function f. We specify the values of these parameters for our specific problem in the Methods section.

2 Background

For our midway report we tested various deep learning baselines, including feedforward networks, LSTMs, temporal convolutional networks, and a simple graph convolutional network, on two standard benchmark datasets in the traffic forecasting literature: (1) METR-LA [8], and (2) PEMS-BAY [12]. For the GCN, we used fixed adjacency matrices provided in the datasets obtained from Ref. [12]. Our test error results are shown in Fig. 1. We did not experiment with complex graph-based architectures on these data sets; however, many of the models we implement in this report are based on architectures originally used on these datasets.

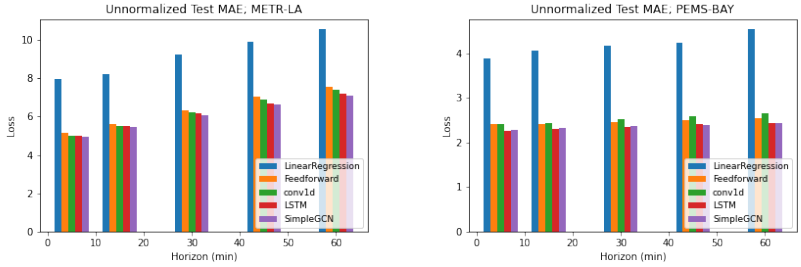

Figure 1: METR-LA and PEMS-BAY test set mean absolute error at multiple prediction horizons.

While road networks are fairly static and well-defined, we were interested in taking the above methods and applying them to a problem where we intuit that there is some spatial correlation, but this correlation is *difficult to specify*. Additionally, we wanted to test how spatio-temporal models from the traffic forecasting literature would perform on similarly structured data in a different domain, which has been less extensively studied.

2.1 Dataset Description

We run our experiments on bike demand data from the New York Citibike dataset [14]. The original dataset contains records of all bike trips between NY Citibike stations from 2013 until the present. We discretize time into 30-minute intervals, and count the number of rentals leaving and entering each station to arrive at demand and supply features for each interval. Since the number of stations has varied over time, and since the trip counts for stations are highly imbalanced, we construct our dataset using the $N = 300$ most popular stations by trip counts from January 2018 through December 2019. We remove all trips with missing station data. A plot of bike demand at the most and least popular stations is shown in Fig. 2. Figure 3 shows the locations of the bike stations with respect to one another. We set $S = H = 10$; at each time t, we use 5 hours (S 30-minute intervals) of history to

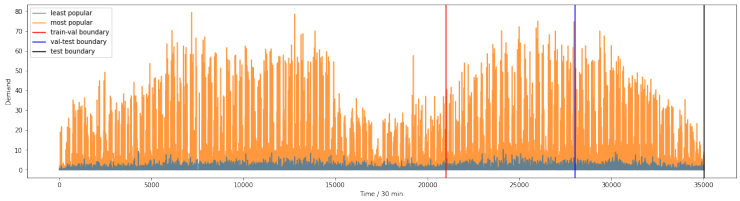

Figure 2: Train-Val-Test Split for Most and Least Popular Stations.

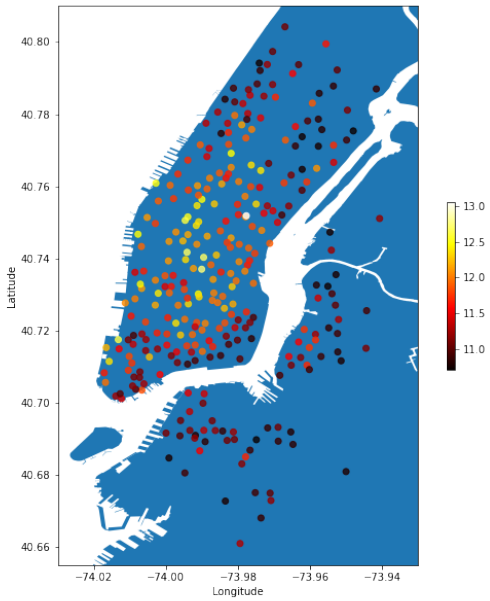

Figure 3: Locations of 300 used stations. Colors based on log of trip counts between 2018-19.

predict the next 5 hours (H 30-minute intervals) of demand and supply at all N stations.We split the data into time-contiguous train, validation and test sets of sizes 60-20-20%. In addition to the demand and supply at each station, we also provide a time-of-day encoding for each datapoint using sine and cosine functions with periods of 1 day. Thus, our input is of dimensions $S \times N \times (d + d_t)$, where $d_t = 2$ is the number of time-based features. This choice of covariates is based on our intuition, but our code can be used with any time-based features or even learnable time features.

Table 1: NY CitiBike processed dataset details.

Dataset	# Nodes	# Features	Horizon	ΔT	# Train	# Val	# Test	# Total
NY Citibike	300	2	10	30 min	21012	7004	7005	35021

3 Related Work

Our work is related to and inspired by work in the traffic forecasting literature, previous work on bike-share demand prediction, as well as work on contextual embeddings.

Traffic forecasting. Yu et. al first introduced graph convolutional networks to the problem of traffic forecasting in Ref. [19]. Their spatio-temporal graph convolutional network (STGCN) architecture,

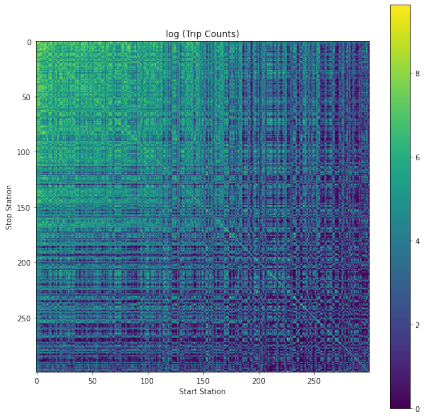

Figure 4: Log Trip Counts

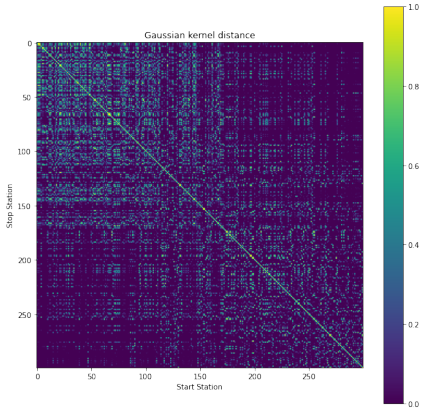

Figure 5: Gaussian kernel distance

which incorporates a GCN layer with a pre-defined adjacency matrices and multiple gated temporal convolutional layers, was used on the METR-LA and PEMS-BAY datasets. We use their architecture both as a baseline and as the basis for our modifications. Our midway report was based upon Ref. [12], which uses recurrent layers instead of convolutional layers to model temporal dependencies. Our baselines were based on those presented in their paper.

Adaptive adjacency matrices. Wu et. al introduced self-adaptive adjacency matrices in Ref. [18]. We use their architecture, GraphWaveNet, as another baseline and another basis for our modifications. Adaptive adjacency matrices have since been applied successfully in many spatio-temporal architectures, including FC-GAGA [15] and AGCRN [3]. Ref. [3] notes that node-agnostic weights in the graph convolutional layer often hurt performance and proposes to embed the nodes into a lower dimensional space, to which a fully-connected layer is then applied. Both of these models use a single node embedding matrix, unlike GraphWaveNet, which uses two.

Bike share prediction. Chai et. al applied multi-graph convolutional networks with encoder-decoder LSTMs to the problem of bike flow prediction on NY Citibike data from 2013 in Ref. [4]. Their work suggests that architectures incorporating information from multiple kinds of graphs (distance-based, correlation-based, and trip-count-based) perform better than those that use any one fixed graph. Ref. [7] also applied graph convolutional networks to bike-share datasets; in addition to building a forecasting model, they apply their work to the problem of optimizing bike rebalancing.

Time-dependent embeddings. Li et al. considers the problem of incorporating time-dependence in continuous-time event-sequence prediction [11]. They propose to augment static event embeddings with time-dependent components to construct time-dependent embeddings. Their ideas could also be incorporated into a more challenging version of the bike flow prediction problem, where time is not discretized into fixed intervals and trips are events that occur in continuous time.

4 Methods

To overcome problems related to *static* and *pre-defined* adjacency matrices, we propose to use *dynamic*, *learnable* adjacency matrices.

4.1 Time-adaptive adjacency matrix

The self-adaptive adjacency matrix used in GraphWaveNet is parametrized in the following way:

$$\mathbf{A} = \sigma(ReLU(\mathbf{E_1}\mathbf{E_2}^T)),$$

where σ is some activation (either sigmoid or tanh) and $\mathbf{E_1}, \mathbf{E_2} \in \mathbb{R}^{N \times d_e}$ are matrices specifying learnable node embeddings of dimension d_e. The ReLU activation is added to promote sparsity of the resulting matrix, while preserving the density of the embeddings. Ref. [3] adapts the above by using $\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{A}$, where the identity matrix is explicitly added to ensure the existence of self-loops. They also propose setting $\mathbf{E_1} = \mathbf{E_2}$ for easier optimization; however, we note that this necessarily

yields a *symmetric* adjacency matrix, which we feel might not be desirable in the context of the bike sharing problem where dependencies may be directional.

To add time-dependence, we loosely build upon the proposed time-contextualized embeddings in Ref. [11]. At each time step t, we modify each embedding matrix above separately as:

$$\tilde{\mathbf{E}}_i = \mathbf{E}_i \odot \sigma\left(\phi_i(\mathbf{E}_i||\mathbf{Z}_t)\right)$$

where $\mathbf{E}_i||\mathbf{Z}_t \in \mathbb{R}^{N\times(d_e+d_t)}$ is a row-wise concatenation of the embedding matrix with the $N \times d_t$ time covariate matrix $\mathbf{Z}_t$ at time t. Note that these covariates may either be fixed and provided with the data or learned; for simplicity, we consider fixed covariates here. ϕ_i is an MLP, whose architecture we keep as a design choice; for our experiments, we use one hidden layer and a fully connected linear output layer. We note that for our problem, since each node is sampled at the same time, we have $\mathbf{Z}_t = [\mathbf{z}_t, ..., \mathbf{z}_t]$, i.e. the time features are the same for each node for any time slice. Also, note that for each input to our model the time-based features have dimensions $S \times N \times d_t$, i.e. we also have variation along the history. This would give us a different $\mathbf{A}$ for each time step in our history. To speed up computation and since we do not expect the adjacency matrix to vary that much, we propose to use some pooling operation along the temporal dimension, e.g. max pooling. We also note that an alternative way to modulate the embeddings may be:

$$\tilde{\mathbf{E}}_i = \mathbf{E}_i + \sigma\left(\phi_i(\mathbf{E}_i||\mathbf{Z}_t)\right)$$

We test both forms in our experiments. Altogether for the input spanning times $t - S$ to t, we use the matrix

$$\mathbf{A} = \mathbf{I} + \sigma(ReLU(\tilde{\mathbf{E}}_1\tilde{\mathbf{E}}_2^T))$$

with $\tilde{\mathbf{E}}_i = \mathbf{E}_i * \sigma\left(\phi_i(\mathbf{E}_i||pool([\mathbf{Z}_{t-S}, ..., \mathbf{Z}_t]))\right)$, where $* = \odot$ or $+$.

4.2 Baselines and Models

We are interested in the comparison between (1) graph-based vs. non-graph based architectures, (2) fixed vs. adaptive adjacency matrices, and (3) dynamic vs. static adjacency matrices. Thus, for our baselines we run: (1) standard linear regression, (2) fully-connected feed-forward network with 2 hidden layers of 256 units (with L2 regularization), (3) fully-connected LSTM network with 2 hidden layers of 32 units, (4) spatio-temporal graph convolutional network (STGCN) [19] with a fixed adjacency matrix based on trip count data as in Ref. [4], and (5) GraphWaveNet with a self-adaptive adjacency matrix. Although designed to be used with a fixed matrix, we also train (6) STGCN with an adaptive matrix. Layer sizes are chosen to ensure that all models are comparably-sized; each of our models uses less than 100,000 parameters, as can be seen in Table 2. All models used here are implemented with TensorFlow 2.8.0. Our code (for both the final and midway reports) is available at https://github.com/vursekar/time-adaptive-gnn.

Table 2: Model Sizes and Training Times (on NVIDIA Tesla K80 GPU).

Model	# Parameters	Approx. Time (s) / Epoch
Linear	820	<2
FNN	81,428	<5
LSTM	90,200	16
STGCN (fixed)	52,884	22
STGCN (adapt)	72,084	22
STGCN (time-adapt)	77,588	23
STGCN (time-adapt-add)	77,588	23
WaveNet (adapt)	78,260	200
WaveNet (time-adapt)	84,468	200
WaveNet (time-adapt-add)	84,468	200

The embedding dimension of adaptive node embeddings is set to 32. STGCN uses multiple blocks which each consist of a GC layer sandwiched between two gated temporal convolutional layers; we use 2 blocks with filter sizes of [64,16,64] and [32,16,32] respectively. Details of the architecture can be found in Ref. [19]. GraphWaveNet also uses multiple blocks, each consisting of temporal and

graph convolutional layers. For all GraphWaveNet experiments, we use the default architecture with 4 blocks and 2 layers per block. All convolutional layers have 32 filters and use a kernel of size 2.

To test our proposed idea, we propose 4 models with time-adaptive matrices: (1) STGCN with multiplicative gating, (2) STGCN with additive gating, (3) GraphWaveNet with multiplicative gating, and (4) GraphWaveNet with additive gating. These models are the same as baselines (5) and (6), except that we add multiplicative or additive time-modulation as described in the previous section.

5 Results

Table 3: Test Losses.

Model	MAE				MSE				SMAPE			
	30 m	2 hr	5 hr	Avg.	30 m	2 hr	5 hr	Avg.	30 min	2 hr	5 hr	Avg.
Linear	0.845	1.261	1.427	1.266	2.495	6.598	8.299	6.661	0.553	0.579	0.601	0.584
FNN	0.777	1.069	1.176	1.075	2.128	4.717	5.658	4.782	**0.518**	0.581	0.594	0.577
LSTM	1.042	0.993	0.987	0.997	4.195	3.692	3.635	3.735	0.573	0.568	**0.568**	0.569
STGCN (fixed)	0.820	1.016	1.118	1.026	2.216	3.983	4.944	4.070	0.533	0.565	0.579	0.565
WaveNet (adapt)	0.746	0.932	1.022	0.937	1.928	3.393	4.130	3.404	0.526	0.566	0.574	0.562
STGCN (adapt)	0.821	0.961	1.082	0.983	2.338	3.572	4.884	3.815	0.532	0.570	0.597	0.572
STGCN (time-adapt)	0.787	0.937	1.048	0.951	2.048	3.380	4.082	3.376	0.527	0.560	0.588	0.563
STGCN (time-adapt-add)	0.789	0.938	1.032	0.945	2.172	3.431	4.053	3.398	**0.518**	0.558	0.578	**0.559**
WaveNet (time-adapt)	0.745	0.908	1.017	0.922	**1.836**	3.063	3.897	3.165	**0.518**	**0.553**	0.598	0.562
WaveNet (time-adapt-add)	**0.744**	**0.903**	**0.986**	**0.909**	1.870	**3.038**	**3.599**	**3.038**	0.521	0.559	0.579	0.560

All models are trained to minimize the mean absolute error across the entire label matrix. A single free GPU on Google Colab (an NVIDIA Tesla K80) was used to train each model. We use the Adam optimizer with an initial learning rate of 0.001 for a maximum of 100 epochs to train all models, except GraphWaveNet, which we train for a maximum of 30 epochs. The learning rate is reduced by a factor of 10 every 20 epochs. The patience for early stopping is set to 10 epochs without decrease in validation loss. For evaluation, we present 3 commonly used metrics: mean absolute error (MAE), mean squared error (MSE), and symmetric mean absolute percentage error (SMAPE). For SMAPE, we follow the convention of not including the factor of 2 in the numerator. Table 3 presents the corresponding test losses for each model at 3 prediction horizons, as well as the average over these horizons. Figure 5 is a visualization of the data presented in the table. In Fig. 5, we show example predictions for some models. For clarity, we only plot demand predictions of the most and least popular stations.

As stated, we compare models along the lines of (1) graph vs. non-graph, (2) fixed vs. learnable adjacency matrix, and (3) static vs. dynamic adjacency matrix.

Looking at the average MAE and MSE, the graph-based architectures seem to perform better than the non-graph based architectures like FNN and LSTM. However, in terms of SMAPE the graph-based architectures perform about the same and sometimes even worse. This may be because the graph-based architectures are learning to reduce the overall loss by favouring the most popular stations. In Table 4 we see that the graph-based architectures indeed perform almost 50% better on average than the non-graph based architectures for the nodes with the most activity. However, the differences are less stark for the less active stations.

STGCN with the adaptive adjacency matrix seems to slightly outperform the version with the fixed adjacency matrix in terms of MAE and MSE. This suggest that a learnable adjacency matrix may sometimes be just as effective as a fixed one; this might be useful in cases where the adjacency matrix is hard to explicitly define.

The models with time-adaptive adjacency matrices do seem to outperform their static counterparts on all counts for both STGCN and GraphWaveNet; the MSE is about 10% lower, however, the difference in SMAPE is quite small. This squares with the intuition that correlations between stations vary as a function of time. Between the two architectures, GraphWaveNet works better with the time-adaptive matrices. Moreover, the additive time modulation seems to work slightly better than the multiplicative

modulation. Perhaps the fact that the multiplicative modulation restricts the norms of the embeddings slightly degrades performance.

Table 4: Avg. Loss at Top 20 and Bottom 20 Stations by Trip Count

Model	MAE		MSE		SMAPE	
	Top 20.	Bottom 20	Top 20	Bottom 20	Top 20	Bottom 20
Linear	2.860	0.549	29.096	0.714	0.528	0.688
FNN	2.280	0.471	20.046	0.641	0.490	0.702
LSTM	1.989	0.424	13.039	0.577	0.456	0.721
STGCN (fixed)	2.065	0.476	16.178	0.631	0.469	0.701
WaveNet (adapt)	1.827	0.424	12.390	0.548	0.444	0.713
STGCN (adapt)	1.995	0.442	14.735	0.618	0.464	0.728
STGCN (time-adapt)	1.844	0.432	11.763	0.586	0.441	0.718
STGCN (time-adapt-add)	1.863	0.421	12.122	0.568	0.440	0.715
WaveNet (time-adapt)	1.768	0.414	10.736	0.561	0.438	0.724
WaveNet (time-adapt-add)	1.727	0.415	10.235	0.543	0.436	0.714

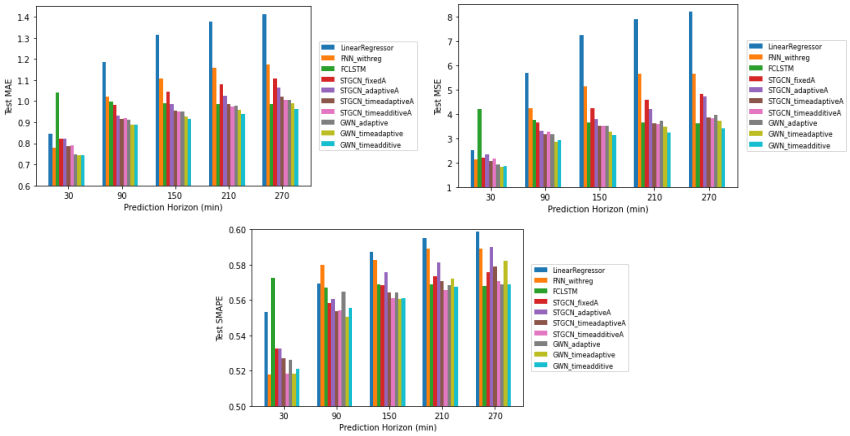

Figure 6: Visualization of test error at different prediction horizons.

6 Discussion and Analysis

Graph vs. non-graph based models. Since the above architectures have already been compared on traffic forecasting datasets, their usefulness is not in question. The performance difference between graph and non-graph based models *on this dataset* remains unclear; while the graph based models outperform the non-graph models in MAE and MSE on average and for most prediction horizons, as the Table 4 results suggest, this might stem from their performance on the most popular stations, which receive more activity. Additionally, the SMAPE values for graph and non-graph based models do not differ by much, which suggests that when we take into account signal magnitude, the performance difference is less stark. One strategy to mitigate/probe this would be to normalize each station's signal independently, e.g. by its maximum. In this way the average performance metric will not be skewed by the subset of stations with more activity than others. Another strategy might be to train and evaluate the models on a subset of stations with more comparable magnitudes.

Fixed vs. adaptive adjacency matrices. STGCN was originally designed to be used with a fixed adjacency matrix. Tables 3 and 4 suggest that on this dataset the architecture performs better or comparably with a self-adaptive matrix. This can mean many things: for example, the fixed adjacency matrix we used did not represent a good spatial prior. Indeed, Ref. [4] showed that a GNN with multiple graphs worked better than one using just trip count-based ones. Nonetheless, as the adjacency matrices plotted in Fig. 2.1 show, it can be quite hard to hand craft a good spatial prior when there are many nodes/features. While we lose interpretability, a learnable matrix that achieves comparable

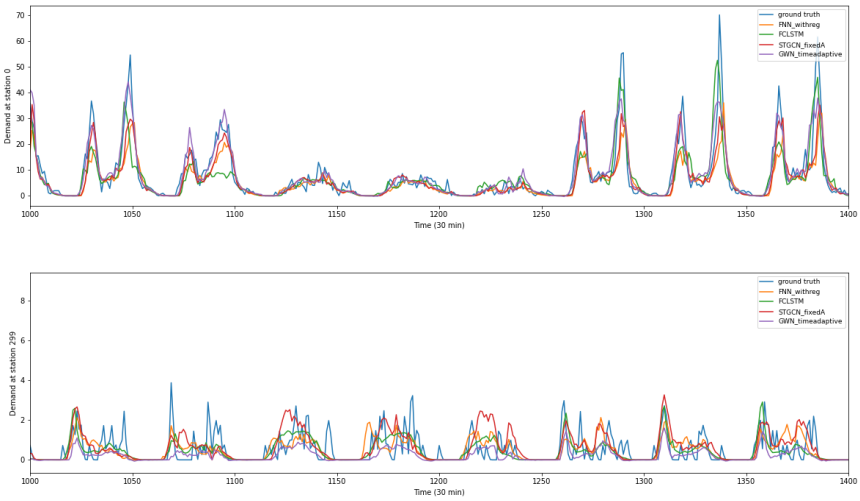

Figure 7: Test predictions on a subset of the test set.

performance might be easier to work with. The most natural next step to probe this design choice would be to run STGCN with the adaptive matrix on the METR-LA and PEMS-BAY datasets and compare its performance with that of the original model.

Static vs. dynamic adjacency matrices. Our results indicate that time-adaptive adjacency matrices outperform their static counterparts on all metrics, including SMAPE, on this dataset. It should be noted, however, that this difference is sometimes small and that the time-adaptive matrices have $\approx 10\%$ more parameters. As a next step, one could run the static and dynamic models on other datasets to make sure that the difference is not an artifact of the Citibike dataset. Additionally, one could reduce the number of parameters for the dynamic models to test if the architecture itself is providing any advantage. Also, we do find a difference in performance between the multiplicative and additive modulations presented in the Methods section. Since these were only two choices here, a natural question then is: what is the "best" way to modulate the embeddings? As mentioned, we apply a pooling operation to the time covariates to reduce computation (and because we suspect that embeddings will not change much for nearby time points); it would be worth investigating if there is a better way to encode these "smoothness" assumptions into the form of the modulation.

References

[1] Bai, L., Yao, L., Kanhere, S. S., Yang, Z., Chu, J. and Wang, X. [2019], Passenger demand forecasting with multi-task convolutional recurrent neural networks, *in* 'Pacific-Asia Conference on Knowledge Discovery and Data Mining', Springer, pp. 29–42.

[2] Bai, L., Yao, L., Kanhere, S., Wang, X., Sheng, Q. et al. [2019], 'Stg2seq: Spatial-temporal graph to sequence model for multi-step passenger demand forecasting', *arXiv preprint arXiv:1905.10069* .

[3] Bai, L., Yao, L., Li, C., Wang, X. and Wang, C. [2020], 'Adaptive graph convolutional recurrent network for traffic forecasting', *Advances in Neural Information Processing Systems* **33**, 17804–17815.

[4] Chai, D., Wang, L. and Yang, Q. [2018], Bike flow prediction with multi-graph convolutional networks, *in* 'Proceedings of the 26th ACM SIGSPATIAL international conference on advances in geographic information systems', pp. 397–400.

[5] Chatfield, C. [2000], *Time-series forecasting*, Chapman and Hall/CRC.

[6] Chen, L., Zhang, D., Wang, L., Yang, D., Ma, X., Li, S., Wu, Z., Pan, G., Nguyen, T.-M.-T. and Jakubowicz, J. [2016], Dynamic cluster-based over-demand prediction in bike sharing systems, *in* 'Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing', pp. 841–852.

[7] Guo, R., Jiang, Z., Huang, J., Tao, J., Wang, C., Li, J. and Chen, L. [2019], Bikenet: Accurate bike demand prediction using graph neural networks for station rebalancing, *in* '2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)', IEEE, pp. 686–693.

[8] Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R. and Shahabi, C. [2014], 'Big data and its technical challenges', *Commun. ACM* **57**(7), 86–94.
URL: *https://doi.org/10.1145/2611567*

[9] Jiang, W. and Luo, J. [2021], 'Graph neural network for traffic forecasting: A survey', *CoRR* **abs/2101.11174**.
URL: *https://arxiv.org/abs/2101.11174*

[10] Kipf, T. N. and Welling, M. [2016], 'Semi-supervised classification with graph convolutional networks', *arXiv preprint arXiv:1609.02907* .

[11] Li, Y., Du, N. and Bengio, S. [2017], 'Time-dependent representation for neural event sequence prediction', *arXiv preprint arXiv:1708.00065* .

[12] Li, Y., Yu, R., Shahabi, C. and Liu, Y. [2018], Diffusion convolutional recurrent neural network: Data-driven traffic forecasting, *in* 'International Conference on Learning Representations'.
URL: *https://openreview.net/forum?id=SJiHXGWAZ*

[13] Li, Y., Zheng, Y., Zhang, H. and Chen, L. [2015], Traffic prediction in a bike-sharing system, *in* 'Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems', pp. 1–10.

[14] Motivate International, I. [n.d.], 'Citi bike system data'.
URL: *https://ride.citibikenyc.com/system-data*

[15] Oreshkin, B. N., Amini, A., Coyle, L. and Coates, M. J. [2021], Fc-gaga: Fully connected gated graph architecture for spatio-temporal traffic forecasting, *in* 'Proc. AAAI Conf. Artificial Intell'.

[16] Sen, R., Yu, H.-F. and Dhillon, I. S. [2019], 'Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting', *Advances in neural information processing systems* **32**.

[17] Wu, Y. and Tan, H. [2016], 'Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework', *arXiv preprint arXiv:1612.01022* .

[18] Wu, Z., Pan, S., Long, G., Jiang, J. and Zhang, C. [2019], 'Graph wavenet for deep spatial-temporal graph modeling', *arXiv preprint arXiv:1906.00121* .

[19] Yu, B., Yin, H. and Zhu, Z. [2018], Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting, *in* 'Proceedings of the 27th International Joint Conference on Artificial Intelligence', IJCAI'18, AAAI Press, p. 3634–3640.

[20] Zheng, C., Fan, X., Wang, C. and Qi, J. [2020], 'Gman: A graph multi-attention network for traffic prediction', *Proceedings of the AAAI Conference on Artificial Intelligence* **34**(01), 1234–1241.
URL: *https://ojs.aaai.org/index.php/AAAI/article/view/5477*