A Study of Joint Graph Inference and Forecasting

Daniel Zügner [1 2] **François-Xavier Aubet** [3] **Victor Garcia Satorras** [4 2] **Tim Januschowski** [3]
Stephan Günnemann [1] **Jan Gasthaus** [3]

Abstract

We study a recent class of models which uses graph neural networks (GNNs) to improve forecasting in multivariate time series. The core assumption behind these models is that there is a latent graph between the time series (nodes) that governs the evolution of the multivariate time series. By parameterizing a graph in a differentiable way, the models aim to improve forecasting quality. We compare four recent models of this class on the forecasting task. Further, we perform ablations to study their behavior under changing conditions, e.g., when disabling the graph-learning modules and providing the ground-truth relations instead. Based on our findings, we propose novel ways of combining the existing architectures.

1. Introduction

Forecasting multivariate time series is a core machine learning task both in science and in industry [19]. Between the individual time series (*nodes*), rich dependencies and interactions (*edges*) govern how the time series evolves. In the simplest case these could be (linear) correlations; other examples include the road network underlying traffic flows [25, 22], or physical relations such as attraction or repulsion affecting trajectories of objects in space [11].

Knowledge of the 'true' relations can be used to make more accurate predictions of how the time series evolves in the future, e.g., by using graph neural networks (GNNs) (e.g., [12, 23, 8, 6, 20, 13, 3, 2]) . Even more, the graph can reveal fundamental insights into the system described by the time series, and may thus be of value in itself, independent of an improvement in the forecasting quality. Therefore, recent works aim at jointly inferring relations between the time series *and* learn to forecast in an end-to-end manner, sometimes without any prior information about the graph [25, 5].

[1]Technical University of Munich [2]Work done while being an intern at AWS AI Labs, Amazon Web Services [3]AWS AI Labs, Amazon Web Services [4]University of Amsterdam. Correspondence to: Daniel Zügner <zuegnerd@in.tum.de>.

Time Series Workshop @ ICML 2021.

Besides potential benefits in forecasting quality, inferring a graph among N time series comes at an inherent computational complexity of $O(N^2)$, which needs to be taken into account when deciding whether to leverage joint graph inference and forecasting. Hence, we consider the following *research questions* in this paper.

(R1) In which scenarios do joint graph inference and forecasting improve forecasting accuracy? Given the diverse domains and settings of multivariate time series forecasting (e.g., underlying spatial relations of sensors in traffic forecasting, sets of sensors measuring different properties of the same system, etc.) it is possible that graph inference helps the forecasting task more in some use cases.

(R2) How do the existing architectures compare in forecasting performance? Are there certain architectural choices that appear beneficial for forecasting?

(R3) What are properties of the inferred graphs by the model? Specifically, how consistent are the inferred graphs across different training runs? How (dis-)similar are the inferred graphs to the "ground-truth" graphs (when known)?

2. Background

Forecasting with Multivariate Time Series

In time series forecasting we are interested in estimating a future series $\mathbf{z}_{t+1:T}$ given its past $\mathbf{z}_{t_0:t}$ and some context information about the past $\mathbf{x}_{t_0:t}$ where variables $t_0 < t < T$ index over time. For the multivariate case, we can consider N time series at a time $\mathbf{z}_{t_0:T} = \{\mathbf{z}_{1,t_0:T}, \ldots, \mathbf{z}_{N,t_0:T}\} \in \mathbb{R}^{N \times T - t_0}$. We model the following conditional distribution:

$$p(\mathbf{z}_{i,t+1:T} | \mathbf{z}_{t_0:t}, \mathbf{x}_{t_0,t}),\ 1 \leq i \leq N, \tag{1}$$

where i indexes over time series. Notice that we are conditioning on all N series in order to estimate the series i.

Time Series Forecasting for graph structured data

When conditioning over multivariate time series as in Eq. (1), we may benefit from modelling the relations between different multivariate time series. An expressive structure to capture such relations are graphs. We can define a graph as a set of nodes $v_i \in \mathcal{V}$ and edges $e_{ij} \in \mathcal{E}$ that relate the nodes. In our case each $\mathbf{z}_i$ is associated to a graph

node v_i. Edges e_{ij} may be given or unkown depending on the dataset, in cases where the underlying graph is latent/unkown we may jointly infer the graph while estimating a forecasting model. In this work we study the performance of a variety of algorithms under different assumptions of the graph structure (known, unkown, partially known). Note that even in the cases where we have "ground-truth" knowledge (e.g., of spatial relations), there may still be additional latent relations which could be discovered by the models.

3. Literature Review

Recent models perform joint graph learning and forecasting in multivariate timeseries. These models are **GTS** ("graph for timeseries") [22], Graph Deviation Network (**GDN**) [5], MTS forecasting with GNNs (**MTGNN**) [25], and Neural Relational Inference (**NRI**) [11]. Here, we briefly introduce these four methods and their differences and commonalities; for a more detailed overview, see Appendix B.

All models can be decomposed into two main components: the **graph learning** and the **forecasting** modules. The former outputs an adjacency matrix describing a graph between the nodes (i.e., timeseries). The latter takes this graph as well as the input timeseries window to forecast the next timestep(s). Once the adjacency matrix has been obtained from the graph learning module, there are many ways of how to leverage it for forecasting the timeseries. The core idea of the models of this study is that the adjacency matrix construction step is differentiable and jointly learned with the forecasting module. Thus, the intuition is that the model will learn graphs which help the forecasting task.

3.1. Graph learning

The goal of the graph learning module is to output an adjacency matrix $\mathbf{A} \in [0,1]^{N \times N}$, where each entry $\mathbf{A}_{ij}$ denotes the edge weight between nodes (i, j). Typically, we aim for $\mathbf{A}$ to be sparse, which reflects the intuition that there are only relatively few useful relations in the latent graph. Each model first represents each node i by a fixed-size vector $\mathbf{h}_i$, followed by a pairwise similarity computation of any pair $\mathbf{h}_i$ and $\mathbf{h}_j$, e.g., by using a fully connected neural network or simply by taking the dot product.

Next, the models obtain the adjacency matrix from the pairwise scores. **MTGNN** and **GDN** do so by taking the K highest scores per node. An advantage of this is that by choosing K appropriately $\mathbf{A}$ is guaranteed to be sparse. On the other hand, the top-K operation is not continuously differentiable, which may pose challenges to end-to-end learning.

NRI and **GTS** first map the pairwise scores into range $[0, 1]$ (e.g., via softmax or sigmoid). The models use the Gumbel softmax trick [15, 10] to sample a discrete adjacency matrix from the edge probabilities in a differentiable way (though gradients are biased); a downside is that we have to take extra steps to obtain a sparse graph, e.g., by regularization.

Moreover, the models can can be broadly split into two groups according to how they compute the fixed-size representations $\mathbf{h}_i$ per node: **MTGNN** and **GDN** simply learn these representations as node embeddings; on the other hand, **NRI** and **GTS** compute the vectors $\mathbf{h}_i$ based on the time series itself. That is, they apply some (shared) function to each timeseries to map it into a fixed-size vector. While **NRI** dynamically produces the representations *per individual window*, **GTS** uses the *whole training timeseries* for each node. The former has the advantage of being more flexible, though more expensive, since we need to compute a $[B \times N \times N]$ tensor to store the *individual* adjacency matrices, where B is the batch size. On the other hand, the graph learned by **GTS** is global, i.e., shared for all time series. It is thus more efficient yet less flexible, as the model cannot adjust the graph for changing inputs during inference time. Moreover, in its current implementation, this leads to **GTS**'s number of parameters growing linearly with the length of the training time series (though this could in principle be resolved via dilated convolutions or pooling).

3.2. Graph-based forecasting

There are many existing models to incorporate graph structure in the forecasting task (e.g., [14, 21, 4, 26, 27, 7, 18, 24]). Each of the models in this study has its own way of forecasting the time series given the input timeseries window and the adjacency matrix constructed by the graph learning module. For instance, **MTGNN** interchanges temporal convolution layers with graph convolution layers, and **GTS** uses a Diffusion-Convolutional Recurrent Neural Network (DCRNN) [14], where the hidden states of each node are diffused via graph convolutions at each timestep. Again, the core idea is that the adjacency matrix used in the graph-based forecasting is itself constructed in a differentiable way and can thus be adjusted by the model to improve forecasting results.

4. Experiments

To address our research questions **(R1)**-**(R3)**, we perform experiments on real-world and synthetic datasets. We repeat all runs five times and report the average; error bars are provided in Table 4 (Appendix).

4.1. Datasets

We briefly describe here the datasets that we use; more details can be found in appendix section C. We scale each timeseries to have zero mean and unit variance or to have range $[0, 1]$ (only SWaT and WADI, as in [5]). For training and evaluation we compute MAE on the *original* scale.

	GTS		MTGNN		NRI		GDN	
	MAE@12	Δ	MAE@12	Δ	MAE@12	Δ	MAE@12	Δ
METR-LA	3.74	+3.27%	3.89	−9.85%	7.8	−7.69%	4.1	+4.87%
PEMS-BAY	1.91	+5.56%	1.97	−6.38%	2.13	-	2.12	−2.31%
Diffusion	0.0684	+3.92%	0.117	−5.24%	0.0614	−44.97%	0.122	−7.88%
DAG	0.695	−0.48%	0.697	−0.40%	0.702	−0.25%	0.692	−3.25%

Table 1. Average forecasting MAE (over five runs) when disabling the graph-learning and forcing the model to use the ground-truth graph. We also show the *percentage change* of the MAE (Δ); e.g., -4% means error is reduced by 4% over the base scenario.

	Random graph								No graph			
	GTS		MTGNN		NRI		GDN		GTS		MTGNN	
	MAE@12	Δ	MAE@12	Δ	MAE@12	Δ	MAE@12	Δ	MAE@12	Δ	MAE@12	Δ
METR-LA	3.77	+4.10%	4.31	−0.03%	7.95	−5.90%	4.05	+3.68%	4.43	+22.20%	4.35	+0.61%
PEMS-BAY	1.86	+2.86%	2.1	+0.02%	2.12	-	2.14	−1.29%	2.15	+18.93%	2.11	+0.51%
WADI	5.97	+0.80%	6.27	−0.09%	7.87	+3.94%	7.2	−4.32%	6.09	+2.89%	6.23	−1.37%
SWaT	0.372	+24.20%	0.684	−4.81%	0.398	−37.94%	0.897	−17.06%	0.574	+91.76%	0.722	+1.26%
Electricity	201.0	+0.71%	185.0	−1.08%	-	-	270.0	−3.70%	205.0	+2.87%	198.0	+5.79%
Solar Energy	2.74	+3.03%	2.71	+0.83%	-	-	2.9	+1.53%	3.01	+13.10%	2.78	+2.96%
Traffic	-	-	-	-	-	-	0.014	+1.92%	-	-	0.0103	−23.51%
Exchange Rate	0.0108	+9.59%	0.0149	+2.99%	0.0121	+16.37%	0.0956	+11.53%	0.0101	+2.14%	0.0145	−15.75%
DAG	0.697	−0.22%	0.7	−0.08%	-	-	0.697	−2.59%	0.698	−0.02%	0.7	−0.11%
Diffusion	0.0703	+6.83%	0.124	+0.43%	-	-	0.132	−0.16%	0.126	+91.70%	0.128	+4.09%

Table 2. Average forecasting MAE (averaged over five runs) when forcing the model to use a (sparse) random graph (left) or when not using a graph at all in the forecasting (right). Relative performance (Δ) as explained in Fig. 1. '-' indicates OOM/timeout after 24 hours.

PEMS-BAY and METR-LA [14] are widely used traffic datasets where we do have knowledge about the underlying graph. To construct the sensor graph, we computed the pairwise road network distances between sensors and build the adjacency matrix using a thresholded Gaussian kernel.

We use a range of other multi-variate datasets for which no graph structure is known: Electricity,[1,2] Solar-energy,[3,2] Exchange-rate[2] and Traffic.[4,2] Further, SWaT[16] and WADI [1] are datasets of sensors measuring water-treatment plants. In the test split there are annotated anomalies where the creators tampered with the water treatment systems. Therefore, SWaT and WADI were originally proposed as anomaly detection datasets (and e.g., used in the GDN paper); however, since the respective training sets are free of anomalies, we use them for our forecasting experiments.

Synthetic datasets. To enhance the real world datasets, we create two synthetic datasets starting with a graph and making sure that the graph has an impact on the connection between the time series. This allows us to speculate that the graph will be of importance for the forecasting of the time series. We create the **Diffusion** dataset by using Personalized PageRank (PPR) [17] to diffuse the multivariate timeseries. We create the **DAG** dataset using a directed acyclic graph (DAG) and making all the children dimensions be a weighted combination of its parents dimensions.

[1] archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

[2] github.com/laiguokun/multivariate-time-series-data

[3] www.nrel.gov/grid/solar-power-data.html

[4] https://pems.dot.ca.gov

4.2. Results

(R1). Here we analyze the forecasting results on the different datasets at horizons 3, 6, and 12, respectively. For reference, we also add a vanilla LSTM [9] baseline that jointly forecasts all timeseries, as well LSTM-U, which consists of N univariate LSTMs. Essentially, the LSTM uses information from *all* timeseries, though lacks typical GNN properties such as permutation equivariance and does not leverage sparsity. LSTM-U is on the other end of the spectrum and simply views all timeseries as completely independent. In Table 4 (appendix) we present the results.

On the popular traffic datasets METR-LA and PEMS-BAY, the GNN models generally dominate the LSTMs. These datasets have known spatial spatial relations among the timeseries, thus this comes as no surprise. NRI's results on METR-LA is quite poor, which we attribute to the relatively large number of nodes and to the fact that the underlying relations are static, while NRI predicts a graph *per window*.

On WADI, interestingly, LSTM-U performs on par with MTGNN. The remaining gap to GTS is relatively small and can potentially be explained by GTS's more sophisticated forecasting procedure. This indicates that on WADI, where we do not have a "straightforward" spatial graph between the nodes, the GNN-based models struggle to find useful relations in the data – or that there are no useful relations in the data to begin with. Similarly, on SWaT, LSTM outperforms all GNN-based models except GTS.

In the synthetic diffusion-based dataset, GTS achieves roughly 50% lower mean absolute error than LSTM. We

attribute this to the fact that the data-generating process (*graph diffusion*) matches well with the DC-RNN architecture used by GTS in the forecasting module. Further, note that on Traffic, GTS ran OOM on a 16GB VRAM GPU for batch size larger than 1, and therefore did not finish within 24 hours. NRI, which is even more computationally expensive, has additional missing values.

In summary, the GNN-based models' edge over the non-graph baselines tends to be largest for datasets with an underlying spatial graph (traffic datasets, Electricity, Solar Energy), and smaller for the datasets where the relations are expected to be more subtle (WADI, SWaT). Future work could compare the GNN-based models to state-of-the-art non-graph forecasting methods in a benchmark study.

(R2). Next we perform ablation experiments on the GNN-based models to study their behavior when removing their graph-learning module. For the forecasting modules, we either provide the ground-truth graph (where known); provide a sparse random graph; or provide no graph. We compare results to the "vanilla" settings of the models, computing the relative change in MAE at horizon 12 in percent.

In Table 1 we show the results for providing the ground-truth graph to the forecasting modules. Strikingly, MTGNN's performance substantially increases, leading to almost 10% less MAE on METR-LA. On PEMS-BAY and METR-LA, MTGNN's results are on par with GTS's. This suggests that MTGNN's forecasting module performs well, and that GTS's graph-learning module may be advantageous. GDN also benefits from ground truth, though the effect is not as pronounced. Interestingly, providing the "true" graph to GTS leads to a slight performance drop on all but one datasets, indicating that the model's graph-learning module is effective at improving forecasting.

In Table 2, we see the results for providing a (sparse) random Erdős Renyi graph to the models (left), or completely disabling the graph processing in the forecasting modules (right). For the random graphs we set the edge probability p such that the expected degree is 30 ($N \geq 100$), 10 ($20 \leq N < 100$), or 3 ($N < 20$). An interesting insight is that for GTS, using a random graph has little or moderate effect on most datasets; and that using no graph at all leads to strong performance drop, indicating that GTS's forecasting module greatly benefits from the sparsity of graphs.

Remarkably, for MTGNN we see relatively little effect when using a random graph or even no graph at all. We hypothesize that this is due to MTGNN's way of constructing the adjacency matrix. It uses kNN-style approach, which has sparse gradients. Further, the edge weights are the result of applying $\tanh$ to the pairwise scores, which may lead to vanishing gradients. Thus, the node embeddings may receive only very little training signal. In contrast, GDN, which also uses node embeddings in the graph learning module, utilizes the node embeddings also in the forecasting task. This may be a way to address the issue of MTGNN. Another approach may be to replace the kNN graph construction with differentiable sampling via the Gumbel softmax trick (as in GTS and NRI). This is an interesting experiment to further investigate whether the strategy of parameterizing the graph based on the time series, employed by NRI and GTS, is generally advantageous over node-embedding-based approaches.

		Avg. corr.	Avg. corr. GT
METR-LA	GDN	0.356	0.212
	MTGNN	-0.001	-0.015
	GTS	0.264	-0.046
	GTS w/ reg.	0.493	0.523
PEMS-BAY	GDN	0.287	0.185
	MTGNN	0.000	-0.008
	GTS	0.164	-0.010
	GTS w/ reg.	0.704	0.684

Table 3. Average correlation of edge scores across different training runs (left), and with the ground-truth graph (right).

(R3). Finally, we measure how consistent the learned edge scores are across training runs as well as how similar the learned adjacency matrices are to the ground truth adjacency matrices. For this we measure the *correlation* of edge scores among re-runs and with the ground-truth graph. Intuitively, high correlation means that the model assigns large/small scores to the same node pairs. A subset of the results is shown in Table 3; see Table 5 (app.) for more details. We can see that (i) for GDN and GTS, the learned adjacency matrices tend to be moderately similar across training runs. Interestingly, only GDN's learned graphs have a nontrivial correlation with the ground truth. This indicates that the models learn *a* graph which is useful for forecasting, which need not have much in common with the "true" (e.g., spatial) graph. Note that for these experiments we have disabled GTS's regularization on the ground-truth graph. When enabling the loss (GTS w/ reg.) we find that, as expected, the learned graphs strongly correlate with the input graph.

5. Conclusion

We present a study of recent models performing joint graph inference and forecasting. We highlight key commonalities and differences among the architectures. In our experiments, we compare the forecasting results of the models and study properties of the different graph-learning modules. For instance, we find MTGNN to be insensitive as to whether the graph-learning module is active or not; though it greatly benefits from access to a ground-truth graph. In general, learning a latent graph is a challenging problem; improvements in terms of expressiveness and computational efficiency could lead to broader applicability. We highlight potential ways of combining the existing architectures.

References

[1] Ahmed, C. M., Palleti, V. R., and Mathur, A. P. Wadi: A water distribution testbed for research in the design of secure cyber physical systems. In *Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks*, CySWATER '17, pp. 25–28, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349758. doi: 10.1145/3055366.3055375.

[2] Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. Netgan: Generating graphs via random walks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 609–618, 2018.

[3] Bojchevski, A., Klicpera, J., Perozzi, B., Kapoor, A., Blais, M., Rózemberczki, B., Lukasik, M., and Günnemann, S. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2020. ACM.

[4] Chen, J., Xu, X., Wu, Y., and Zheng, H. Gc-lstm: Graph convolution embedded lstm for dynamic link prediction. *arXiv preprint arXiv:1812.04206*, 2018.

[5] Deng, A. and Hooi, B. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada*, pp. 2–9, 2021.

[6] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 06–11 Aug 2017.

[7] Guo, S., Lin, Y., Feng, N., Song, C., and Wan, H. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 922–929, 2019.

[8] Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[9] Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. 2017.

[11] Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pp. 2688–2697. PMLR, 2018.

[12] Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[13] Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*, 2019.

[14] Li, Y., Yu, R., Shahabi, C., and Liu, Y. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018.

[15] Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *International Conference on Learning Representations*, 2017.

[16] Mathur, A. P. and Tippenhauer, N. O. Swat: a water treatment testbed for research and training on ics security. In *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*, pp. 31–36, 2016. doi: 10.1109/CySWater.2016.7469060.

[17] Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[18] Panagopoulos, G., Nikolentzos, G., and Vazirgiannis, M. Transfer graph neural networks for pandemic forecasting. 2020.

[19] Petropoulos, F., Apiletti, D., Assimakopoulos, V., Babai, M. Z., Barrow, D. K., Bergmeir, C., Bessa, R. J., Boylan, J. E., Browell, J., Carnevale, C., Castle, J. L., Cirillo, P., Clements, M. P., Cordeiro, C., Oliveira, F. L. C., Baets, S. D., Dokumentov, A., Fiszeder, P., Franses, P. H., Gilliland, M., Gönül, M. S., Goodwin, P., Grossi, L., Grushka-Cockayne, Y., Guidolin, M., Guidolin, M., Gunter, U., Guo, X., Guseo, R., Harvey, N., Hendry, D. F., Hollyman, R., Januschowski, T., Jeon, J., Jose, V. R. R., Kang, Y., Koehler, A. B., Kolassa, S., Kourentzes, N., Leva, S., Li, F., Litsiou, K., Makridakis, S., Martinez, A. B., Meeran, S., Modis,

T., Nikolopoulos, K., Önkal, D., Paccagnini, A., Panapakidis, I., Pavía, J. M., Pedio, M., Pedregal, D. J., Pinson, P., Ramos, P., Rapach, D. E., Reade, J. J., Rostami-Tabar, B., Rubaszek, M., Sermpinis, G., Shang, H. L., Spiliotis, E., Syntetos, A. A., Talagala, P. D., Talagala, T. S., Tashman, L., Thomakos, D., Thorarinsdottir, T., Todini, E., Arenas, J. R. T., Wang, X., Winkler, R. L., Yusupova, A., and Ziel, F. Forecasting: theory and practice. *arXiv preprint arXiv:2012.03854*, 2020.

[20] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

[21] Seo, Y., Defferrard, M., Vandergheynst, P., and Bresson, X. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing (ICONIP)*, 2017.

[22] Shang, C., Chen, J., and Bi, J. Discrete graph structure learning for forecasting multiple time series. In *International Conference on Learning Representations*, 2021.

[23] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[24] Wang, B., Luo, X., Zhang, F., Yuan, B., Bertozzi, A. L., and Brantingham, P. J. Graph-Based Deep Modeling and Real Time Forecasting of Sparse Spatio-Temporal Data. *arXiv e-prints*, April 2018.

[25] Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X., and Zhang, C. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 753–763, 2020.

[26] Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., and Li, H. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9): 3848–3858, 2020. doi: 10.1109/TITS.2019.2935152.

[27] Zhu, J., Song, Y., Zhao, L., and Li, H. A3t-gcn: attention temporal graph convolutional network for traffic forecasting. *arXiv preprint arXiv:2006.11583*, 2020.

Dataset	Model	MAE @ 3	MAE @ 6	MAE @ 12
METR-LA	LSTM	3.4950 ± 0.0104	3.7121 ± 0.0117	4.1049 ± 0.0113
	LSTM-U	$3.4164 \pm -$	$4.0916 \pm -$	$5.1411 \pm -$
	NRI	4.6802 ± 0.0813	6.3878 ± 0.0873	8.4661 ± 0.0985
	GDN	3.1490 ± 0.0165	3.4818 ± 0.0144	3.9096 ± 0.0123
	MTGNN	3.0160 ± 0.0044	3.5738 ± 0.0047	4.3076 ± 0.0057
	GTS	$\mathbf{2.8840} \pm 0.0052$	$\mathbf{3.2695} \pm 0.0062$	$\mathbf{3.7013} \pm 0.0055$
PEMS-BAY	LSTM	2.0428 ± 0.0045	2.1110 ± 0.0045	2.2425 ± 0.0071
	GDN	1.8900 ± 0.0095	2.0214 ± 0.0097	2.1722 ± 0.0119
	MTGNN	1.3189 ± 0.0017	1.6904 ± 0.0020	2.1007 ± 0.0039
	GTS	$\mathbf{1.2677} \pm 0.0002$	$\mathbf{1.5551} \pm 0.0012$	$\mathbf{1.8133} \pm 0.0034$
WADI	LSTM	6.6584 ± 0.0558	6.7547 ± 0.0651	6.7887 ± 0.0406
	LSTM-U	5.9805 ± 0.0362	6.1059 ± 0.0230	6.3512 ± 0.0524
	NRI	$6.8025 \pm -$	$7.0759 \pm -$	$7.5703 \pm -$
	GDN	7.4096 ± 0.3068	7.4425 ± 0.2210	7.5232 ± 0.2815
	MTGNN	5.9537 ± 0.0381	6.1083 ± 0.0417	6.2535 ± 0.0293
	GTS	$\mathbf{5.4742} \pm 0.0072$	$\mathbf{5.5754} \pm 0.0094$	$\mathbf{5.7715} \pm 0.0080$
SWaT	LSTM	0.3000 ± 0.0133	0.3296 ± 0.0105	0.4324 ± 0.0163
	LSTM-U	0.2869 ± 0.0012	0.4868 ± 0.0008	0.8826 ± 0.0011
	NRI	0.4147 ± 0.0139	0.4798 ± 0.0142	0.6408 ± 0.0110
	GDN	0.8029 ± 0.0447	0.8537 ± 0.0689	1.0812 ± 0.1495
	MTGNN	0.4878 ± 0.0105	0.5371 ± 0.0161	0.7040 ± 0.0247
	GTS	$\mathbf{0.2420} \pm 0.0382$	$\mathbf{0.2792} \pm 0.0391$	$\mathbf{0.3874} \pm 0.0435$
Electricity	LSTM	323.3455 ± 3.8540	384.2395 ± 10.7886	352.4884 ± 4.2168
	LSTM-U	710.9171 ± 0.7324	1079.3943 ± 2.9989	849.2502 ± 1.8949
	GDN	265.1665 ± 3.0810	269.2244 ± 2.3668	280.4004 ± 1.4468
	MTGNN	$\mathbf{170.1549} \pm 2.9775$	186.0044 ± 4.7505	$\mathbf{193.4988} \pm 4.4654$
	GTS	175.8778 ± 1.0221	$\mathbf{185.7905} \pm 0.9314$	199.5826 ± 1.3747
Solar Energy	LSTM	1.9820 ± 0.0157	2.6776 ± 0.0213	4.2408 ± 0.0239
	LSTM-U	2.7603 ± 0.0071	4.3665 ± 0.0027	6.2828 ± 0.0024
	GDN	2.0953 ± 0.0181	2.3299 ± 0.0212	2.8556 ± 0.0403
	MTGNN	1.5117 ± 0.0049	2.0513 ± 0.0076	2.6889 ± 0.0133
	GTS	$\mathbf{1.4199} \pm 0.0040$	$\mathbf{1.9260} \pm 0.0128$	$\mathbf{2.6577} \pm 0.0329$
Traffic	LSTM	0.0157 ± 0.0002	0.0178 ± 0.0003	0.0173 ± 0.0003
	LSTM-U	0.0285 ± 0.0001	0.0332 ± 0.0002	0.0289 ± 0.0001
	GDN	0.0132 ± 0.0001	0.0134 ± 0.0001	0.0137 ± 0.0001
	MTGNN	$\mathbf{0.0102} \pm 0.0003$	$\mathbf{0.0107} \pm 0.0004$	$\mathbf{0.0108} \pm 0.0003$
Exchange Rate	LSTM	0.0141 ± 0.0013	0.0187 ± 0.0020	0.0190 ± 0.0018
	LSTM-U	0.0057 ± 0.0002	0.0076 ± 0.0001	0.0102 ± 0.0001
	NRI	$\mathbf{0.0047} \pm 0.0001$	0.0073 ± 0.0002	0.0111 ± 0.0005
	MTGNN	0.0109 ± 0.0023	0.0146 ± 0.0033	0.0136 ± 0.0013
	GTS	0.0047 ± 0.0000	$\mathbf{0.0070} \pm 0.0000$	$\mathbf{0.0099} \pm 0.0000$
DAG	LSTM	0.6976 ± 0.0028	0.7079 ± 0.0030	0.7507 ± 0.0026
	LSTM-U	0.7154 ± 0.0007	0.7278 ± 0.0007	0.7816 ± 0.0007
	NRI	0.6132 ± 0.0005	0.6297 ± 0.0009	0.7034 ± 0.0013
	GDN	0.6363 ± 0.0015	0.6519 ± 0.0014	0.7154 ± 0.0011
	MTGNN	0.6107 ± 0.0005	0.6277 ± 0.0009	0.6999 ± 0.0010
	GTS	$\mathbf{0.6088} \pm 0.0003$	$\mathbf{0.6254} \pm 0.0004$	$\mathbf{0.6960} \pm 0.0004$
Diffusion	LSTM	0.1064 ± 0.0001	0.1355 ± 0.0002	0.1405 ± 0.0003
	LSTM-U	0.0986 ± 0.0000	0.1333 ± 0.0000	0.1393 ± 0.0001
	NRI	$0.0704 \pm -$	$0.0894 \pm -$	$0.1120 \pm -$
	GDN	0.0890 ± 0.0005	0.1109 ± 0.0003	0.1325 ± 0.0003
	MTGNN	0.0739 ± 0.0002	0.0969 ± 0.0003	0.1239 ± 0.0008
	GTS	$\mathbf{0.0620} \pm 0.0001$	$\mathbf{0.0655} \pm 0.0002$	$\mathbf{0.0706} \pm 0.0003$

Table 4. Results overview. $\pm$ indicates standard error of the mean (SEM) over five runs. Missing rows indicate OOM/timeout after 24 hours. '$-$' for SEM means that only one run finished witin 24 hours.

A. Additional results

In Table 4, we provide the results on the forecasting task. In Table 5, we provide additional correlation results of the learned graphs.

B. Model details

B.1. Graph for Time Series (GTS)

GTS ("graph for time series") [22] is a recent model aiming to jointly learn a latent graph in the time series and use it for MTS forecasting. The model consists of two main components: graph learning, and graph-based forecasting.

Graph learning. The graph learning module first maps the training partition of each time series $\mathbf{z}_i$ to a fixed-size vector representation $\mathbf{h}_i$ via a 1D-convolutional neural network. Then, all pairs $(\mathbf{h}_i, \mathbf{h}_j)$ are processed by an MLP to output the probability of an edge between i and j.

$$\mathbf{h}_i = f_{\text{FC}}\left(\text{Vec}\left(f_{\text{conv}}(\mathbf{z}_i)\right)\right), \quad \theta_{ij} = \sigma(g_{\text{FC}}\left([\mathbf{h}_i||\mathbf{h}_j]\right)),$$

where $\sigma(\cdot)$ denotes the logistic sigmoid function and $||$ denotes vector concatenation. Finally, a discrete adjacency matrix $\mathbf{A}$ is obtained via element-wise, differentiable sampling, i.e., $\mathbf{A}_{ij} \sim \text{Ber}(\theta_{ij})$, using the Gumbel softmax trick [10, 15]. Note that GTS uses the complete training partition to parameterize the adjacency matrix at each batch. This means that (i) the model can use information from the whole (training) time series at training time; (ii) the model cannot adjust the learned graph at test time; (iii) the number of parameters grows linearly with the length of the input timeseries, which could be improved by adding dilation or pooling to the convolutional encoder.

Forecasting. The forecasting module uses the graph produced by the graph learning module, a Diffusion-Convolutional RNN (DCRNN) [14]. A DCRNN essentially updates hidden states collectively for all series via a graph convolution, which replaces the usual multiplication with a weight matrix. Thus, *at each time step*, the RNN uses the learnt graph to locally average the hidden representation of timeseries in their graph neighborhood:

$$\begin{aligned}
\mathbf{R}_t &= \sigma\left(\text{GNN}_R\left([\mathbf{Z}_t||\mathbf{H}_{t-1}];\mathbf{A}\right) + b_R\right)\\
\mathbf{C}_t &= \tanh\left(\text{GNN}_C\left([\mathbf{Z}_t||\mathbf{R}_t \odot \mathbf{H}_{t-1}];\mathbf{A}\right) + b_C\right)\\
\mathbf{U}_t &= \sigma\left(\text{GNN}_U\left([\mathbf{Z}_t||\mathbf{H}_{t-1}];\mathbf{A}\right) + b_U\right)\\
\mathbf{H}_t &= \mathbf{U}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{U}_t) \odot \mathbf{C}_t
\end{aligned}$$

$$\text{GNN}(\mathbf{Z};\mathbf{A}) = \sum_{k=1}^{K} \left(\mathbf{D}_O^{-1}\mathbf{A}\right)^k \mathbf{Z}\mathbf{W}_O^{(k)} + \left(\mathbf{D}_I^{-1}\mathbf{A}\right)^k \mathbf{Z}\mathbf{W}_I^{(k)}$$

where $\mathbf{D}_O$ and $\mathbf{D}_I$ are diagonal matrices whose entries are the out- and in-degrees of the nodes in $\mathbf{A}$, respectively; $\mathbf{W}_O^{(k)}$ and $\mathbf{W}_I^{(k)}$, $1 \le k \le K$ are learnable weight matrices. GTS uses $K = 2$.

Model	Dataset	Avg. corr.	Avg. corr. GT
NRI	METR-LA	0.44	-0.24
	WADI	-0.11	n/a
	SWaT	0.10	n/a
	Exchange Rate	0.68	n/a
	DAG	0.64	-0.00
	Diffusion	0.85	0.02
GDN	METR-LA	0.36	0.21
	PEMS-BAY	0.29	0.19
	WADI	0.13	n/a
	SWaT	0.25	n/a
	Electricity	0.17	n/a
	Solar Energy	0.15	n/a
	Traffic	0.10	n/a
	Exchange Rate	0.44	n/a
	DAG	0.22	0.04
	Diffusion	0.68	0.00
MTGNN	METR-LA	-0.00	-0.01
	PEMS-BAY	-0.00	-0.01
	WADI	0.00	n/a
	SWaT	0.01	n/a
	Electricity	0.00	n/a
	Solar Energy	0.00	n/a
	Traffic	-0.00	n/a
	Exchange Rate	0.15	n/a
	DAG	0.00	0.00
	Diffusion	-0.00	0.00
GTS	METR-LA	0.26	-0.05
	PEMS-BAY	0.16	-0.01
	WADI	0.46	n/a
	SWaT	0.51	n/a
	Solar Energy	0.02	n/a
	Exchange Rate	0.38	n/a
	DAG	0.46	0.02
	Diffusion	0.01	-0.00

Table 5. Average correlation of edge scores among different training runs (left), and average correlation of the resulting edge scores with the ground-truth graph (where available).

Regularization. The authors propose to incorporate potential a-priori knowledge about the ground-truth graph via a regularization loss in the form of element-wise binary cross entropy loss between the learned and prior graph.

B.2. MTS Forecasting with GNNs (MTGNN)

Like GTS, MTGNN [25] also consists of a graph learning and forecasting module, though the implementations of these modules are different.

Graph learning. In contrast to GTS, which parameterizes the representation $\mathbf{h}_i$ of a timeseries using a neural network based on the input timeseries, MTGNN learns two embedding vectors per node (i.e., timeseries), i.e., two embedding matrices $\mathbf{E}_1, \mathbf{E}_2$. Pairwise scores $\mathbf{A}_{ij}$ are computed as

$$\begin{aligned}\mathbf{M}_1 &= \tanh(\alpha \mathbf{E}_1 \mathbf{W}_1) \\ \mathbf{M}_2 &= \tanh(\alpha \mathbf{E}_2 \mathbf{W}_2) \\ \mathbf{A} &= \mathrm{ReLU}\left(\tanh\left(\alpha\left(\mathbf{M}_1\mathbf{M}_2^T - \mathbf{M}_2\mathbf{M}_1^T\right)\right)\right),\end{aligned}$$

where the formulation of $\mathbf{A}$ ensures that it is asymmetric, i.e., if $\mathbf{A}_{ij}$ is positive, $\mathbf{A}_{ji}$ is zero. Finally, only the top K scores per row $\mathbf{A}_i$ are kept to ensure sparsity of the adjacency matrix. The node embeddings are trained in an end-to-end fashion on the forecasting task.

Forecasting. The main difference to the RNN-based forecasting module in GTS is that MTGNN uses temporal convolutions combined with graph convolution layers. MTGNN stacks three blocks of interchanging inception-style temporal convolution layers and graph convolution layers to predict the next timestep(s) of the timeseries.

B.3. Graph Deviation Network (GDN)

Graph Deviation Network (GDN) [5] is a recent model aimed at *anomaly detection* in multivariate timeseries. The model is trained on MTS forecasting, and anomalies are flagged when the predicted value deviates strongly from the observed value. Since the model is essentially a forecasting method by construction, we chose to include it as a baseline in this work.

Graph learning. Similar to MTGNN, GDN infers the graph by learning a node embedding $\mathbf{v}_i$ per node. Specifically, the model builds a k-NN graph where the similarity metric is the cosine of a pair of embeddings.

Forecasting. The forecasting module is based on the Graph Attention Network (GAT) [23] architecture.

$$\mathbf{h}_i = \mathrm{ReLU}\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{ij} \mathbf{W} \mathbf{z}_j\right),$$

where $\mathbf{z}_j \in \mathbb{R}^w$ is the input timeseries window of node j, $\mathbf{W}$ is a learned weight matrix, and α_{ij} are attention scores, computed as follows.

$$\alpha_{ij} = \mathrm{softmax}_j\left(\mathrm{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{g}_i || \mathbf{g}_j\right]\right)\right),$$

where $\mathbf{g}_i = [\mathbf{v}_i || \mathbf{W}\mathbf{z}_i]$, and $\mathbf{a}$ is a learned vector. The next predicted value(s) for all nodes are predicted jointly by a stack of fully connected layers f_{MLP}:

$$\hat{\mathbf{s}} \in \mathbb{R}^{N \times \hat{T}} = f_{\text{MLP}}\left(\left[\mathbf{v}_1 \odot \mathbf{h}_1 || \mathbf{v}_2 \odot \mathbf{h}_2 || \ldots || \mathbf{v}_N \odot \mathbf{h}_N\right]\right),$$

where $\hat{T}$ is the number of predicted timesteps.

B.4. Neural Relational Inference (NRI)

The Neural Relational Inference (NRI) [11] model assumes a slightly different setting than GTS, MTGNN, and GDN. Instead of learning a *global, static* graph over the whole time series, NRI infers a graph *per input window*. While this setting is more flexible, it comes at the drawback of having inherent $O(B \times N^2)$ memory complexity, where B is the batch size. Thus, NRI can only realistically scale to small graphs, i.e. at most $N \approx 50$. NRI is a VAE-based architecture consisting of an encoder module predicting edge probabilities and a decoder module performing the forecasting.

Graph learning. The encoder module interchanges neural networks on the node and edge representations, respectively:

$$\begin{aligned}\mathbf{h}_j^{(1)} &= f_{\text{emb}}(\mathbf{z}_j) & \mathbf{h}_{(i,j)}^{(1)} &= f_{\text{e}}^{(1)}([\mathbf{h}_i^{(1)} || \mathbf{h}_j^{(1)}]) \\ \mathbf{h}_j^{(2)} &= f_{\text{v}}^{(1)}(\sum_{i \neq j} \mathbf{h}_{(i,j)}^{(1)}) & \mathbf{h}_{(i,j)}^{(2)} &= f_{\text{e}}^{(2)}([\mathbf{h}_i^{(2)} || \mathbf{h}_j^{(2)}])\end{aligned}$$

Finally, the edge type posterior is $q_\phi(\mathbf{z}_{(\mathbf{i},\mathbf{j})} | \mathbf{z}_{\mathbf{t_0}:\mathbf{t}}) = \mathrm{softmax}(\mathbf{h}_{(i,j)}^{(2)})$, where one edge type can be hard-coded to denote 'no edge'. Similar to GTS, NRI samples a discrete graph using the Gumbel softmax trick.

Forecasting. The decoder module is similar to the encoder. It has a fully connected neural network per edge type, which processes the respective input pairs of nodes connected by the specific edge type. Finally, for each node, the representations of incoming edges are aggregated, and a final neural network predicts the value of the next timestep. For the decoder, the authors propose an MLP-based and a RNN-based variant.

C. Datasets

We describe here more in details the characteristics of the datasets used.

In Table 6 we summarize the real-world datasets used in this work. SWaT[16] and WADI [1] are datasets of sensors measuring water-treatment plants. In the test split there are annotated anoalies where the creators tampered with

Dataset	# Samples	N	Ground truth?
PEMS-BAY [14]	52,116	325	Yes
METR-LA [14]	34,272	207	Yes
WADI[5] [1]	1,187,951	122	No
SWAT[5] [16]	475,200	51	No
Electricity[6,7]	26,304	321	No
Solar-energy[8,7]	52,560	137	No
Exchange-rate[7]	7,588	8	No
Traffic[9,7]	17,544	862	No

Table 6. Real-world dataset summary.

the water treatment systems. Therefore, SWaT and WADI were originally proposed as anomaly detection datasets (and e.g., used in the GDN paper); however, since the respective training sets are free of anomalies, we use them for our forecasting experiments.

PEMS-BAY and METR-LA [14] are widely used traffic datasets where we do have knowledge about the underlying graph. To construct the sensor graph, the computed the pairwise road network distances between sensors and build the adjacency matrix using a thresholded Gaussian kernel.

C.1. Synthetic datasets

One drawback about real-world datasets – even the traffic datasets for which we have some knowledge about the relations – is that we do not know the true data-generating process and how the graph interacts with it. To address this, we generated two synthetic datasets where relations between nodes are handcrafted. An advantage of this is that we know in advance the true dependencies between nodes.

Diffusion-based dataset. For each of the N timeseries, we first randomly sample parameters of a sinusoidal function, i.e., its frequency, amplitude, horizontal, and vertical shift. Next, we partition the nodes into K clusters and generate an undirected graph from a Stochastic Blockmodel (SBM), such that nodes within a cluster are more densely connected than between clusters. We use Personalized PageRank (PPR) [17] to diffuse the multivariate timeseries, i.e., compute for each timeseries the weighted combination of itself and the other nodes in its vicinity. For each node, we add independent Gaussian noise to the other nodes before averaging. Thus far we have induced correlation between nodes in the same cluster. Finally, we perform a weighted combination of the timeseries *before* and *after* diffusion, where the diffused timeseries is lagged by C timesteps. This means, each timestep $\mathbf{z}_t = \alpha \cdot \tilde{\mathbf{z}}_t + (1-\alpha) \cdot \hat{\mathbf{z}}_{t-C}$. This way, knowing the value of i's neighbors C steps is useful to predict its current value, rewarding models which have correctly identified the relations.

DAG-based dataset. As an alternative, we generate a dataset based on a directed acyclic graph (DAG). We induce an arbitrary topological order defined by the node IDs, i.e., $1, \ldots, N$. We iterate over nodes in increasing topological order. For node i, we randomly sample *incoming* edges from all nodes $j < i$ with uniform probability p. If no incoming edges were sampled for i, we generate its timeseries as a random sinusoidal function as described above and add Gaussian noise. Otherwise, i's timeseries is a randomly weighted combination of modified timeseries j for which (i, j) is an edge and $j < i$. We modify the input timeseries by applying random horizontal and vertical shift and stretch, and add some noise again. Thus, i's values are directly determined by its incoming edges (except for some noise), and we expect models which correctly identify the relations to perform well in the forecasting task.

For the diffusion dataset we set $\alpha = 0.75$, $C = 10$, and the restart probability in the PPR computation to 0.15. We choose $K = 5$ clusters; the edge probability within clusters is 0.5, and between clusters we have 0.05.

For DAG, the edge probability $p = 0.1$. For both synthetic datasets, we set $N = 100$.

D. Training details

Generally, we use the hyperparameters provided by the authors of the respective papers. We train all models on a horizon of 12 timesteps, where the training loss is the mean absolute error of all 12 time steps. For all datasets except SWaT and WADI, we ignore targets with value zero (as in [22, 25]), as these correspond to missing values. We train models for a maximum of 200 epochs and use early stopping with patience of 20 epochs. We use the validation MAE for early stopping. For SWaT and WADI as well as DAG and Diffusion, we fix the window length to 20. For METR-LA and PEMS-BAY, we set the window length to 12, as proposed by [25, 22]. For Electricity, Solar Energy, Traffic, and Exchange Rate, window size is 168 as in [25].

[5] As proposed by [5], we subsample SWaT and WADI by a factor of 10 in the time dimension using the median operation.

[6] archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

[7] github.com/laiguokun/multivariate-time-series-data

[8] www.nrel.gov/grid/solar-power-data.html

[9] https://pems.dot.ca.gov