Multivariate Time Series Forecasting with Latent Graph Inference

Victor Garcia Satorras
University of Amsterdam
Amsterdam, The Netherlands

Syama Sundar Rangapuram
Amazon
Berlin, Germany

Tim Januschowski
Zalando
Berlin, Germany

Abstract

This paper introduces a new approach for Multivariate Time Series forecasting that jointly infers and leverages relations among time series. Its modularity allows it to be integrated with current univariate methods. Our approach allows to trade-off accuracy and computational efficiency gradually via offering on one extreme inference of a potentially fully-connected graph or on another extreme a bipartite graph. In the potentially fully-connected case we consider all pairwise interactions among time-series which yields the best forecasting accuracy. Conversely, the bipartite case leverages the dependency structure by inter-communicating the N time series through a small set of K auxiliary nodes that we introduce. This reduces the time and memory complexity w.r.t. previous graph inference methods from $O(N^2)$ to $O(NK)$ with a small trade-off in accuracy. We demonstrate the effectiveness of our model in a variety of datasets where both of its variants perform better or very competitively to previous graph inference methods in terms of forecasting accuracy and time efficiency.

1. Introduction

Time Series Forecasting (TSF) has been widely studied due to its practical significance in a variety of applications such as climate modelling (Mudelsee, 2019), supply chain management in retail (Böse et al., 2017), market analysis in finance (Andersen et al., 2005), traffic control (Li et al., 2017) and medicine (Kaushik et al., 2020) among others. In TSF, given a sequence of data points indexed over time, we aim to estimate its future values based on previously observed data. Data is often multivariate, meaning that multiple variables vary over time, each variable may not only depend on its own historical values, but also on other variables’ past. Efficiently modelling the dependencies among these variables is still an open problem.

Multivariate Time Series (MTS) methods aim to leverage dependencies between variables in order to improve the forecasting accuracy. A natural way to model non-linear dependencies in deep learning is via Graph Neural Networks (GNNs) (Bruna et al., 2013; Defferrard et al., 2016; Kipf & Welling, 2016). In fact, GNNs have been successfully applied to MTS forecasting (Chen et al., 2020a; Li et al., 2017; Yu et al., 2017), leveraging the relations among different series. But these methods require a pre-defined adjacency matrix which may only be available in some specific datasets, for example, traffic datasets, where it can be constructed from the spatial structure of a city. More recently, a family of methods that do not require a pre-defined adjacency matrix have been proposed (Franceschi et al., 2019; Shang et al., 2021; Wu et al., 2019, 2020). In this case, a latent graph representation is inferred while forecasting, allowing to operate on a larger variety of datasets. Our work belongs to this category.

However, inferring all pairwise relations may come at a higher computational cost, as the number of relations scales quadratically $O(N^2)$ w.r.t. the number of nodes/time series. This limits the scalability to large datasets. Additionally, the above mentioned latent graph inference methods perform message passing updates at every time step iteration which is also expensive.

To overcome these limitations, we propose a new latent graph inference algorithm for MTS forecasting that is more efficient than previous algorithms while achieving better or competitive performance. We cast the latent graph inference as a modular and easy-to-implement extension to current univariate models. The graph is dynamically inferred for each time series input allowing a more flexible representation than a static graph for the whole dataset. Additionally, we can reduce the complexity from $O(N^2)$ (Fully Connected Assumption) to $O(NK)$ (Bipartite Assumption) where $K \ll N$ for a small trade off in performance.

1Work done while at Amazon. Correspondence to: Victor Garcia Satorras, v.garciasatorras@uva.nl
2. Background

2.1. Time Series Forecasting

In time series forecasting, we aim to estimate a future time series $\mathbf{x}_{t+1:T}$ given its past $\mathbf{x}_{t_0:t}$ where $t_0 \leq t \leq T$ indexes over time, and (optionally) some context information $c$. For the multivariate case, we assume the time series is composed of $N$ variates at a time such that $\mathbf{x}_{t_0:T} = \{\mathbf{x}_{1,t_0:T}, \ldots, \mathbf{x}_{N,t_0:T}\} \in \mathbb{R}^{N \times T - t_0 + 1}$. In this section we distinguish two main categories of time series forecasting methods, Global Univariate and Multivariate.

**Global Univariate methods**: In this case, we only use the past of each univariate to predict its future. However, the model weights $\theta_u$ are shared across all univariate time series. More formally:

$$\hat{x}_{i,t+1:T} = f_u(x_{i,t_0:t}, c; \theta_u) \quad (1)$$

where $\hat{x}$ denotes the estimated values, $i \in \{1, \ldots, N\}$ indexes over variates and $f_u(\cdot)$ is the estimator function with learnable parameters $\theta_u$ shared across time series. Conditioning on the past of each univariate may limit the performance of the forecasting algorithm compared to multivariate ones. Despite that, it simplifies the design of $f_u$ and already provides reasonable results. A popular example of a global univariate model is DeepAR (Salinas et al., 2020).

**Multivariate methods**: Multivariate methods condition on all past data (all $N$ variates) and directly predict the multivariate target. More formally:

$$\hat{x}_{t+1:T} = f_m(x_{t_0:t}, c). \quad (2)$$

Different variables may be correlated and/or depend on the same con-founders. For example, in retail forecasting, PPE masks and antibacterial soaps jointly increased in demand during the early days of the COVID-19 pandemic. In traffic forecasting, an increase of the outcome traffic flow in a given neighborhood may result in an increase of the income traffic flow on another one. Modelling these dependencies may improve the forecasting accuracy, but it may come at a cost of higher complexity and hence more expensive algorithms, especially when trying to model all pairwise interactions between variates.

2.2. Graph Neural Networks

Graph Neural Networks (GNNs) (Bruna et al., 2013; Defferrard et al., 2016; Kipf & Welling, 2016) operate directly on graph structured data. They have gained a lot of attention in the last years due to their success in a large variety of domains which benefit from modelling interactions between different nodes/entities. In the context of multivariate time series, GNNs can be used to model the interactions between time series. In this work we consider the type of GNN introduced by (Gilmer et al., 2017). Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $v_i \in \mathcal{V}$ and edges $e_{ij} \in \mathcal{E}$, we define a graph convolutional layer as:

$$m_{ij} = \phi_e(h_i^l, h_j^l), \quad h_{i}^{l+1} = \phi_h(h_i^l, \sum_{j \in \mathcal{N}(i)} \alpha_{ij} m_{ij}) \quad (3)$$

Where $\phi_e$ and $\phi_h$ are the edge and node functions, usually approximated as Multi Layer Perceptrons (MLPs), $h_i^l \in \mathbb{R}^d$ is the $n_f$-dimensional embedding of a node $v_i$ at layer $l$ and $m_{ij}$ is the edge embedding that propagates information from node $v_j$ to $v_i$. A GNN is constructed by stacking multiple of these Graph Convolutional Layers $h_i^{l+1} = \text{GCL}[h_i^l, \mathcal{E}]$. Additionally, in (3) we include $\alpha_{ij} \in (0, 1)$ which is a scalar value that performs the edge inference or attention over the neighbors similarly to Velickovic et al. (2017). Following (Satorras et al., 2021), we choose this value to be computed as the output of a function $\alpha_{ij} = \phi_\alpha(m_{ij})$ where $\phi_\alpha$ is composed of just a linear layer followed by a sigmoid activation function.

3. Related Work

Time series forecasting has been extensively studied in the past due to its practical significance with a number of recent overview articles available (Benidis et al., 2020; Lim & Zohren, 2021; Petropoulos et al., 2022). Traditionally, most classical methods are univariate in nature (see e.g., Hyndman & Athanasopoulos (2017) for an overview). While some of these have multi-variate extensions (e.g., ARMA and VARMA models), they are limited by the amount of related time series information they can incorporate. Dynamic factor models (Geweke, 1977; Wang et al., 2019a) are fore-runners of a family of models that has recently received more attention, the so-called global models (Januschowski et al., 2019; Montero-Manso & Hyndman, 2022), see Section 2.1. These global models estimate their parameters over an entire panel of time series, but still produce a univariate forecast. Many such global models have been proposed building on neural network architectures like RNNs (Bander, et al., 2019; Liberty et al., 2020; Salinas et al., 2020), CNNs (Chen et al., 2020b; Wen et al., 2017), Transformers (Eisenach et al., 2020; Li et al., 2019; Lim et al., 2021) and also combining classical probabilistic models with deep learning (de B´ezenac et al., 2020; Kurle et al., 2020; Rangapuram et al., 2018). However, unlike our method, these global models do not explicitly model the relationship between the time series in the panel. Most recently, global multi-variate forecasting models have received attention, in particular models that attempt to capture the relationship of the time series via a multi-variate likelihood (de B´ezenac et al., 2020; Rasul et al., 2020; 2021; Salinas et al., 2019). A complementary approach consists of capturing the multi-variate nature of many modern forecasting problems primarily by using a multi-variate time series as input. For this, a natural way to model and exploit the
relationship between time series is via GNNs (Bruna et al., 2013; Defferrard et al., 2016; Kipf & Welling, 2016). Even in those scenarios where the adjacency of the graph is explicitly provided, attention or a latent graph can be inferred from the node embeddings such that GNNs can still leverage the structure of the data. Some examples of latent graph inference or attention are (Wang et al., 2019b) in point clouds, (Franceschi et al., 2019) in semi-supervised graph classification, (Ying et al., 2018) in hierarchical graph representation learning, (Kipf et al., 2018) in modelling dynamical systems, (Kazi et al., 2020) in zero-shot learning and 3D point cloud segmentation, (Garcia & Bruna, 2017; Kossen et al., 2021) in image classification, (Cranmer et al., 2020) in inferring symbolic representations and (Fuchs et al., 2020; Satorras et al., 2021) in molecular property prediction. In MTS forecasting, (Lai et al., 2018; Shih et al., 2019) provide some of the first deep learning approaches designed to leverage pair-wise dependencies among time series. More recent methods (Li et al., 2017; Seo et al., 2018; Yu et al., 2017; Zhao et al., 2019) are built in the intersection of GNNs and time series forecasting but they require a pre-defined adjacency matrix. Lately, new methods that infer a latent graph from the node embeddings have been introduced in MTS forecasting (Cao et al., 2021; Shang et al., 2021; Wu et al., 2020), and in MTS anomaly detection (Deng & Hooi, 2021; Zhang et al., 2020). These methods can be applied to any dataset even when there is not an explicitly defined adjacency matrix. The main limitation is the computational cost, which occurs in the edge inference and scales quadratically $O(N^2)$ w.r.t the number of nodes/time series $N$ or $O(N^3)$ in (Cao et al., 2021; Wu et al., 2020).

Our approach is related to (Shang et al., 2021; Wu et al., 2020), but in contrast (i) our latent graph is dynamically inferred for each input instead of static for the whole dataset; (ii) it only requires the message exchange at time step $t$ which renders the graph operation cheaper and modular; and (iii) it optionally reduces the number of edges from $O(N^2)$ to $O(NK)$ when using a bipartite assumption ($K \ll N$).

### Linear vs Non-linear message passing

A shared potential limitation among the mentioned methods that use GNNs for MTS Forecasting (Cao et al., 2021; Li et al., 2017; Seo et al., 2018; Shang et al., 2021; Wu et al., 2019, 2020; Yu et al., 2017; Zhao et al., 2019) is the use of a linear function in the edge operation (Defferrard et al., 2016; Kipf & Welling, 2016). In contrast, we allow for non-linearities in our model. Table 1 provides an overview comparing these two alternatives by writing the linear graph convolutional layer in the same format as the non-linear one (Gilmer et al., 2017) from Section 2.2. Where $\theta_c, \theta_h \in \mathbb{R}^{d \times d}$ are learnable parameters.

The linear graph convolutional layer is commonly written in matrix form as we show in Appendix A.3. Note that while the non-linear alternative approximates the edge operation with an MLP $\phi_e$, in the linear case the message operation among two nodes $(i, j)$ only depends on node $h_j$ and it is agnostic to the identity of the receiver node $i$. This can be problematic when inferring relations among two nodes since the message operation only depends on one of them. Therefore, when no adjacency matrix is provided, an attention mechanism that depends on both $i, j$ can address some of these limitations $m_{ij} = \alpha(i, j) h_j^T$. However, note that in the non-linear case, $\phi_e$ is a universal approximator that can be interpreted as a generalization of the linear case with attention $\phi_e(h_i, h_j) = \alpha(i, j) h_j^T$. In practice, even if the non-linear case can more flexibly model $m_{ij}$, it has been shown in previous literature that attention or edge inference (Satorras et al., 2021; Vaswani et al., 2017) can be a strong inductive bias which in our case is able to infer the graph structure as we will show in the experiments section.

### 4. Method

In Section 2.1 we discussed two families of forecasting methods: Global Univariate Models and Multivariate Models. We cast our multivariate algorithm as a modular extension of the univariate case from Table 1. We can break down the Univariate Model in two main steps $f_{enc} = f_{enc} \circ f_{dec}$ such that $x_{i,t}, t \xrightarrow{f_{enc}} z_i, h_i, \hat{x}_{i,t+1:T}$, where $f_{enc}$ encodes the input signal $x_{i,t}, t$ (and optionally some context information $c_i$) into an embedding $z_i$, and $f_{dec}$ estimates the future signal from this embedding. In our method, we include a multivariate aggregation module $AGG$ in between $f_{enc}$ and $f_{dec}$ that propagates information among nodes in the latent space $z = \{z_1, \ldots, z_N\}$. This aggregation module takes as input the embedding $z_i = f_{enc}(x_{i,t}, c_i)$, and outputs a modified embedding $\tilde{z} = AGG(z)$ where information has been propagated among nodes. These new embeddings are then passed as input to the decoder $f_{dec}$. The resulting algorithm is:

| Operation | Univariate Encoder | Multivariate extension | Univariate Decoder |
|-----------|--------------------|------------------------|--------------------|
| $z_i$     | $f_{enc}(x_{i,t}, c_i)$ | $\tilde{z}$ = $AGG(z)$ | $\hat{x}_{i,t+1:T}$ = $f_{dec}(\tilde{z})$ |

Table 1. Non-linear vs Linear Graph Conv. Layer comparison.
our model does not propagate information among nodes at every time step $[t_0, \ldots, t]$ but only in the AGG module. This makes the algorithm cheaper since the message propagation step is usually an expensive operation. Additionally, we experimented including a unique identifier of the time series in the encoded signal $z_i$ as context information $c_i = \text{id}$, which resulted in a significant improvement in accuracy as we will show in the experiments section. In the following subsections we propose two different GNN configurations for the AGG module depending on the assumed structure of the inferred graph. One is a potentially fully connected graph and the other is a bipartite graph that communicates the $N$ time series through a small set of $K$ auxiliary nodes that we introduce, each has its benefits and weaknesses regarding performance and scalability. An illustration of the whole algorithm is presented in Figure 1.

**Fully connected graph assumption — FC-GNN**

From the two configurations, this is the most straightforward to implement given a standard GNN. It results in good performance, but its complexity scales $O(N^2)$ with respect to the number of nodes $N$. Despite this, it still resulted in faster computation times than previous $O(N^2)$ methods. We can start by defining a fully connected graph $G = \{V, E\}$ where all nodes exchange messages among each other such that $e_{ij} = 1$ for all $e_{ij} \in E$. Each time series embedding $z_i$ obtained from the Univariate Encoder $f_{\text{enc}}$ is associated with a node of the graph $v_i \in V$. Then, we can directly use the GNN defined in Section 2.2 as the aggregation module AGG where each embedding $z_i$ is provided as the input $h_0^i$ to the GNN $f_{\text{AGG}}$, and after the GNN runs for $L$ layers, the output node embedding $h_L^i$ is provided as the input $\hat{z}_i$ to the Univariate Decoder $f_{\text{dec}}$. Moreover, despite the fully connected assumption, the GNN infers attention weights $\alpha_{ij} \in (0, 1)$ for each edge and input sample that "gate" the exchanged messages $m_i = \sum_{i \neq j} \alpha_{ij} m_{ij}$. This can be interpreted as dynamically inferring the graph. The reason is that in GNNs we can write the message aggregation as $m_i = \sum_{j \in N(i)} m_{ij} = \sum_{i \neq j} e_{ij} m_{ij}$ where $e_{ij}$ is 1 if the edge exists or 0 otherwise. We can see that the following expressions $\sum_{i \neq j} e_{ij} m_{ij} \approx \sum_{i \neq j} \alpha_{ij} m_{ij}$ becomes equivalent when the soft estimation $\alpha_{ij} = \phi_0(m_{ij})$ approximates/infers the underlying graph $e_{ij}$ structure from the data. In the experiments section, we name this model a Fully Connected Graph Neural Network (FC-GNN).

**Bipartite graph assumption — BP-GNN**

In the previous section we presented a GNN method that exchanges information among time series under a fully connected graph assumption which computationally scales $O(N^2)$. In this section we introduce a bipartite graph assumption (BP-GNN) which reduces the complexity to $O(NK)$, $K$ being a parameter of choice $K \ll N$. To accomplish this, we define a bipartite graph $G = \{\mathcal{V}, \mathcal{U}, \mathcal{E}\}$, where $\mathcal{V}$ is a set of $N$ nodes corresponding to the $N$ time series and $\mathcal{U}$ is a set of $K$ auxiliary nodes. Nodes $\mathcal{V}$ have associated the time series embeddings $z = \{z_1, \ldots, z_N\}$ and auxiliary nodes $\mathcal{U}$ have associated the embeddings $u = \{u_1, \ldots, u_K\}$ which are free learnable parameters initialized as gaussian. Edges $\mathcal{E}$ interconnect all nodes between the two subsets $\{\mathcal{V}, \mathcal{U}\}$, but there are no connections among nodes within the same subset. This results in $2NK$ edges.

The message passing scheme works in the following way. We input into the GNN the union of the two node subsets $\mathcal{V} = \mathcal{V} \cup \mathcal{U}$. Specifically, the input embedding $h^0$ defined in Equation 3 is the concatenation of the time series embeddings $z$ with the auxiliary node embeddings $u$ (i.e. $h^0 = [z|u]$). Then, messages follow an asynchronous schedule, first information is propagated from the times series nodes to the auxiliary nodes $\mathcal{V} \rightarrow \mathcal{U}$, next the other way around $\mathcal{U} \rightarrow \mathcal{V}$.

We have conceptually defined the Bipartite Graph Neural Network. In Table 2 we introduce the equations that formally define it as an extension of the standard GNN Equation 3.
Notice that it can be simply formulated as a two steps process where the indexes $i,j$ belong to each one of the subsets $U$ or $Y$ depending on the direction of the messages ($Y \rightarrow U$ or $U \rightarrow Y$). Additionally, we used different learnable parameters between Step 1 and Step 2 in the modules $\phi_e$, $\phi_a$ and $\phi_h$ since it resulted in better performance than sharing parameters. Following, we define the adjacency matrices corresponding to the two message passing steps (assuming all inference parameters $\alpha_{ij} = 1$):

\[
A_1 = \begin{bmatrix} 0_{N \times N} & 0_{N \times K} \\ 1_{K \times N} & 0_{K \times K} \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0_{N \times N} & 1_{N \times K} \\ 0_{K \times N} & 0_{K \times K} \end{bmatrix} \tag{7}
\]

$A_1$ refers to $Y \rightarrow U$ and $A_2$ refers to $U \rightarrow Y$. The product of these two matrices $\tilde{A} = A_2A_1$ (Appendix B) defines the sum of all paths that communicate the time series nodes $Y$ among each other through the auxiliary nodes $U$. Notice that for $K \geq 0$ all nodes $\bar{Y}$ can potentially communicate among them being more efficient than FC-GNN as long as $0 \leq K < N/2$.

**Architecture details**

Recall that our method is composed of three main modules, the encoder $f_{\text{enc}}$, the decoder $f_{\text{dec}}$ and the aggregation module AGG. We choose to use relatively simple networks as encoder and decoder. The decoder $f_{\text{dec}}$ is defined as a Multi Layer Perceptron (MLP) with a single hidden layer and a residual connection in all experiments. The encoder $f_{\text{enc}}$ is also defined as an MLP for METR-LA, PEMS-BAY and our synthetic datasets and as a Convolutional Neural Network (CNN) for the other datasets since these require a larger encoding length and the translation equivariance of CNNs showed to be more beneficial. The encoder $f_{\text{enc}}$, first encodes the input signal $x_{i,t:0}$ to an embedding vector by using the mentioned MLP or CNN, and then concatenates a unique identifier ($c_i = \text{id}$) to the obtained embedding vector resulting in $z_i$. $c_i$ could (optionally) include additional context information if it was provided in the dataset. The combination of these simple networks with our proposed aggregation module AGG fully defines our model.

The aggregation module AGG was defined under two different assumptions, the Fully Connected Graph assumption (FC-GNN) and the Bipartite Graph assumption (BG-GNN). In both cases the GNN is fully parametrized by the networks $\phi_e$, $\phi_h$ and $\phi_a$. $\phi_e$ consists of a two layers MLP, $\phi_h$ is a one layer MLP with a skip connection from he input to the output and $\phi_a$ is just a linear layer followed by a Sigmoid activation function. All architecture choices are explained in more detail in Appendix A.2. We optimized the Mean Absolute Error as the loss $\mathcal{L} = \| (\hat{x}_{i+1:T}, x_{i+1:T}) \|$ for training.

**5. Experiments**

**5.1. Datasets and Baselines**

| Dataset    | #Nodes | # Samples | Context length | Pred. length |
|------------|--------|-----------|----------------|--------------|
| METR-LA    | 207    | 34,272    | 12             | 12           |
| PEMS-BAY   | 325    | 52,116    | 12             | 12           |
| Solar-Energy | 137    | 52,560    | 168            | 1            |
| Traffic    | 862    | 17,544    | 168            | 1            |
| Electricity | 321    | 26,304    | 168            | 1            |
| Exchange-Rate | 8     | 7,588     | 168            | 1            |

We first evaluate our method on METR-LA and PEMS-BAY datasets (Li et al., 2017) which record traffic speed statistics on the highways of Los Angeles county and the Bay Area respectively. We also consider the publicly available Solar-Energy, Traffic, Electricity and Exchange-Rate data sets. Specifications for each dataset are presented in Table 3 and further detailed in Appendices B and C. We compare to the following three main types of baselines:

- **Univariate**: In this case a single model is trained for all time series but they are treated independently without message exchange among them. These baselines include a simple linear Auto Regressive model (AR) and a variation of our FC-GNN method that we called NE-GNN where all edges have been removed such that the aggregation function $AGG$ becomes equivalent to a multilayer perceptron defined by $\phi_h$.

- **Multivariate with a known graph**: These methods require a previously defined graph, therefore they are restricted to those datasets where an adjacency matrix can be predefined (e.g. METR-LA, PEMS-BAY). From this group we compare to DCRNN (Li et al., 2017), STGCN (Yu et al., 2017) and MRA-BGCN (Chen et al., 2020a).

- **Multivariate with graph inference or attention**: These methods exchange information among different time series by simultaneously inferring relations among them or by attention mechanisms. From this group we compare to LDS (Franceschi et al., 2019), LST-Skip (Lai et al., 2018), TPA-LSTM (Shih et al., 2019), MTGNN (Wu et al., 2020) and GTS (Shang et al., 2021). Graph WaveNet (Wu et al., 2019) also belongs to this group but unlike the others it jointly uses a pre-defined adjacency matrix. Comparisons to NRI (Kipf et al., 2018) and NRE (Kipf et al., 2018) can be found in previous literature (Shang et al., 2021, Zügner et al., 2021). We additionally include a variation of our FC-GNN without...
Table 4. Benchmark on METR-LA and PEMS-BAY datasets. Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE) are reported for different time horizons \{15, 30, 60\} minutes. Results have been averaged over 5 runs.

|                  | METR-LA | PEMS-BAY |
|------------------|---------|----------|
|                  | 15 min  | 30 min   | 60 min |
|                  | 15 min  | 30 min   | 60 min |
| x5 Linear / AR   | 3.81    | 8.80     | 9.13\% |
| DCRNN            | 3.06    | 6.30     | 8.24\% |
| STGCN            | 4.14    | 8.64     | 12.70\%|
| MRA-BGCN         | 1.72    | 3.91     | 3.93\% |
| Graph WaveNet*   | 6.30    | 12.91    | 16.72\%|
| LDS              | 2.07    | 4.79     | 5.04\% |
| MTGNN            | 2.75    | 5.35     | 7.10\% |
| GTS              | 2.15    | 4.87     | 4.77\% |
| Our models       | 3.06    | 6.17     | 8.30\% |
| NE-GNN (w/o id)  | 1.63    | 3.70     | 3.49\% |
| FC-GNN (w/o id)  | 3.49    | 7.23     | 9.87\% |
| NE-GNN           | 1.94    | 4.49     | 1.33\% |
| NE-GNN (K=4)    | 1.94    | 4.49     | 1.33\% |
| MTGNN            | 2.69    | 5.15     | 6.90\% |
| GTS              | 2.64    | 5.19     | 6.79\% |

**Table 5.** Forward time in seconds for different methods.

| Method              | Forward Time (s) METR-LA | PEMS-BAY |
|---------------------|--------------------------|----------|
| Linear              | .0002                    | .0002    |
| DCRNN               | .2599                    | .2754    |
| Graph WaveNet       | .0500                    | .0673    |
| MTGNN               | .0160                    | .0371    |
| GTS                 | .0869                    | .1087    |
| NE-GNN              | .0033                    | .0047    |
| FC-GNN              | .0108                    | .0253    |
| BP-GNN (K=4)        | .0044                    | .0046    |

unique node ids, we denote it by (w/o id) beside the model name. GTS numbers have been obtained by averaging over 3 runs its official implementation.

### 5.2. Main results

In this section we evaluate our method in METR-LA and PEMS-BAY datasets. For this experiment we used the training setup from GTS (Shang et al. 2021) which uses the dataset partitions and evaluation metrics originally proposed in (Li et al. 2017). All our models have been trained by minimizing the Mean Absolute Error (MAE) between the predicted and ground truth samples. The reported metrics are MAE, Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE) from (Li et al., 2017). All our models have been averaged over 5 runs.

Our results show that FC-GNN is more efficient than previous methods in both METR-LA and PEMS-BAY by a large margin. FC-GNN is also more efficient than previous methods in both datasets but it is still limited by the $O(N^2)$ scalability. In the next Section 5.3 we will see that BP-GNN becomes even more efficient for larger graphs compared to the other methods thanks to its better scalability.
A graph can be pre-defined, therefore, methods that rely on a known graph are not directly applicable. To ensure comparability, we use the same training settings as [Lai et al., 2018; Shih et al., 2019; Wu et al., 2020] in which the network is trained to predict one time step into the future (single step forecasting) with a given horizon (3, 6, 12, or 24) by minimizing the Mean Absolute Error (MAE). All datasets have been split in 60%/20%/20% for training/val/test respectively. As proposed in [Lai et al., 2018] we use the Root Relative Squared Error (RSE) and Empirical Correlation Coefficient (CORR) as evaluation metrics (both defined in Appendix C.2). All our models (FC-GNN, BP-GNN and NE-GNN) contain 2 graph convolutional layers and 128 features in the hidden layers. The timing results have been obtained with batch size 16. Further implementation details are provided in Appendix C.3. Results with standard deviations are reported for different horizons {3, 6, 12, 24}. All results have been averaged over 5 runs. #Top2 column counts how many metrics in each row are in the top 2 (i.e. bold).

### Results

| Methods | Metrics | #Top2 | Solar-Energy | Traffic | Electricity | Exchange-Rate |
|---------|---------|-------|--------------|---------|-------------|---------------|
| AR      | RSE     | (0)   | .2435 .3790 | .5991   | .0995       | .0228         |
| RNN-GRU | CORR    | (0)   | .9710 .9263 | .7752   | .8845       | .9734         |
| RNN-GRU | CORR    | (0)   | .1843 .2559 | .4777   | .0864       | .0226         |
| LST-skip| CORR    | (0)   | .9843 .9690 | .8721   | .9283       | .9735         |
| TPA-LSTM| CORR    | (0)   | .9843 .9690 | .8721   | .9283       | .9735         |
| MTGNN   | RSE     | (5)   | .9521 .9404 | .0907   | .0174       | .9790         |
| NE-GNN  | RSE     | (2)   | .1898 .2580 | .4212   | .0762       | .0175         |
| FC-GNN  | RSE     | (12)  | .1651 .2202 | .4057   | .0732       | .0174         |
| BP-GNN  | RSE     | (11)  | .9876 .9765 | .9024   | .0740       | .0175         |
| BP-GNN  | RSE     | (10)  | .9865 .9751 | .8999   | .9191       | .9796         |
| BP-GNN  | CORR    | (0)   | .9191 .9396 | .8744   | .9948       | .9796         |
| BP-GNN  | CORR    | (0)   | .9191 .9396 | .8744   | .9948       | .9796         |

Table 6. Benchmark on Solar-Energy, Traffic, Electricity and Exchange-Rate. Root Relative Squared Error (RSE) and Empirical Correlation Coefficient (CORR) are reported for different horizons {3, 6, 12, 24}. All results have been averaged over 5 runs. #Top2 column counts how many metrics in each row are in the top 2 (i.e. bold).
Figure 2. Inferred adjacency matrices and MAE losses in the proposed synthetic datasets for FC-GNN, BP-GNN and NE-GNN.

$$N(\beta x_{i-1} \mod N, t-5; \sigma^2)$$, where $$\beta = 0.9$$ and $$\sigma = 0.5^2$$.

(b) We name the second dataset "Correlated Sinusoids", inspired by the Discrete Sine Transformation we generate arbitrary signals as the sum of different sinusoids plus gaussian noise. The gaussian noise is independently added to each time series $$i$$ but the same sinusoid can be shared among different nodes creating strong dependencies. We define different clusters of nodes based on their dependencies, as an example, the first Correlated Sinusoids column in Figure 2 has $$N = 10$$ nodes and two clusters of 5 nodes each $$C = \{5, 5\}$$, the dataset in the last column has $$N = 250$$ nodes with clusters ranging from size 1 to 22 such that $$C = \{1, \ldots, 22\}$$. Further details and visualizations of these two datasets are provided in Appendix D.1. Both BP-GNN and FC-GNN consist of a single graph convolution from which adjacencies are obtained. We forecast the next time step into the future and optimize the MAE during training. Since our graph inference mechanism is dynamic we average them over 10 timesteps $$t$$. Further implementation details are described in Appendix D.2.

Results are reported in Figure 2 for different synthetic datasets which nodes range from $$N = 10$$ to $$N = 250$$. FC-GNN perfectly infers the ground truth adjacencies in all reported datasets and it also achieves the lowest MAE test loss. For the inferred adjacency matrices in BP-GNN, it is harder to develop a visual intuition given the auxiliary nodes, but the same sinusoid can be shared among different nodes creating strong dependencies. We define different clusters of nodes based on their dependencies, as an example, the first Correlated Sinusoids column in Figure 2 has $$N = 10$$ nodes and two clusters of 5 nodes each $$C = \{5, 5\}$$, the dataset in the last column has $$N = 250$$ nodes with clusters ranging from size 1 to 22 such that $$C = \{1, \ldots, 22\}$$. Further details and visualizations of these two datasets are provided in Appendix D.1. Both BP-GNN and FC-GNN consist of a single graph convolution from which adjacencies are obtained. We forecast the next time step into the future and optimize the MAE during training. Since our graph inference mechanism is dynamic we average them over 10 timesteps $$t$$. Further implementation details are described in Appendix D.2.

Figure 3. Left: MAE in METR-LA when sweeping $$K$$ from 0 to 32 auxiliary nodes (10 runs average). Right: MAE in Cycle Graph $$N=10$$, when sweeping $$K$$ from 1 to 10 (4 runs average).

5.5. Choosing the number of auxiliary nodes $$K$$

In all previous sections we set a relatively small number of auxiliary nodes ($$K = 4$$) for BP-GNN. In this section we analyze the BP-GNN performance as we sweep over different $$K$$ values (Figure 3). We noticed that for sparser graph datasets as "Cycle Graph", increasing $$K$$ benefits more than in less sparse graph datasets. In practice, in previous experiments, we chose $$K$$ to be relatively small since in real world datasets as METR-LA, small $$K$$ values already provide competitive accuracy with a good trade-off in complexity.

5.6. Complexity analysis

In Figure 4 we provide a visual comparison of the scalability of FC-GNN and BP-GNN ($K=4$). The right plot shows the vast difference in the number of edges between both methods as we increase the number of nodes. The left plot shows how this translates to the forward time in seconds in the experiments from Section 5.3.

Figure 4. Left: Forward time in seconds for FC-GNN and BP-GNN on different datasets. Right: Scalability of the number of edges w.r.t. the number of nodes $$N$$.

6. Conclusions

We presented a novel approach for multi-variate time series forecasting which lends itself to easy integration into existing univariate approaches by ways of a graph neural network (GNN) component. This GNN infers a latent graph in the space of the embeddings of the univariate time series and can be inserted in many neural-network based forecasting models. We show that this additional graph/dependency structure improves forecasting accuracy both in general and in particular when compared to the state of the art. We alleviate typical scalability concerns for GNNs via allowing the introduction of auxiliary nodes in the latent graph which we construct
as a bipartite graph that bounds the computational complexity. We show that this leads to computationally favorable properties as expected while the trade-off with forecasting accuracy is manageable.

References
Andersen, T. G., Bollerslev, T., Christoffersen, P., and Diebold, F. X. Volatility forecasting, 2005.

Bandara, K., Shi, P., Bergmeir, C., Hewamalage, H., Tran, Q., and Seaman, B. Sales demand forecast in e-commerce using a long short-term memory neural network methodology. In International Conference on Neural Information Processing, pp. 462–474. Springer, 2019.

Benidis, K., Rangapuram, S. S., Flunkert, V., Wang, B., Maddix, D. C., Türkmen, A. C., Gasthaus, J., Bohlke-Schneider, M., Salinas, D., Stella, L., Callot, L., and Januschowski, T. Neural forecasting: Introduction and literature overview. CoRR, abs/2004.10240, 2020.

Böse, J.-H., Flunkert, V., Gasthaus, J., Januschowski, T., Lange, D., Salinas, D., Schelter, S., Seeger, M., and Wang, Y. Probabilistic demand forecasting at scale. Proceedings of the VLDB Endowment, 10(12):1694–1705, 2017.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203, 2013.

Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., et al. Spectral temporal graph neural network for multivariate time-series forecasting. arXiv preprint arXiv:2103.07719, 2021.

Chen, W., Chen, L., Xie, Y., Cao, W., Gao, Y., and Feng, X. Multi-range attentive bicomponent graph convolutional network for traffic forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pp. 3529–3536, 2020a.

Chen, Y., Kang, Y., Chen, Y., and Wang, Z. Probabilistic forecasting with temporal convolutional neural network. Neurocomputing, 399:491–501, 2020b.

Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. Discovering symbolic models from deep learning with inductive biases. arXiv preprint arXiv:2006.11287, 2020.

de Bezenac, E., Rangapuram, S. S., Benidis, K., Bohlke-Schneider, M., Kurele, R., Stella, L., Hasson, H., Gallinari, P., and Januschowski, T. Normalizing kalman filters for multivariate time series analysis. Advances in Neural Information Processing Systems, 33, 2020.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in neural information processing systems, pp. 3844–3852, 2016.

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.

Eisenach, C., Patel, Y., and Madeka, D. Mttrans: Multi-horizon forecasts with context dependent and feedback-aware attention. arXiv preprint arXiv:2009.14799, 2020.

Franceschi, L., Niepert, M., Pontil, M., and He, X. Learning discrete structures for graph neural networks. In International conference on machine learning, pp. 1972–1982. PMLR, 2019.

Fuchs, F. B., Worrall, D. E., Fischer, V., and Welling, M. Se (3)-transformers: 3d roto-translation equivariant attention networks. arXiv preprint arXiv:2006.10503, 2020.

Garcia, V. and Bruna, J. Few-shot learning with graph neural networks. arXiv preprint arXiv:1711.04043, 2017.

Geweke, J. The dynamic factor analysis of economic time series. Latent variables in socio-economic models, 1977.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In International conference on machine learning, pp. 1263–1272. PMLR, 2017.

Hyndman, R. J. and Athanasopoulos, G. Forecasting: Principles and practice. www.otexts.org/fpp, 987507109, 2017.

Januschowski, T., Gasthaus, J., Wang, Y., Salinas, D., Flunkert, V., Bohlke-Schneider, M., and Callot, L. Criteria for classifying forecasting methods. International Journal of Forecasting, 2019.

Kaushik, S., Choudhury, A., Sheron, P. K., Dasgupta, N., Natarajan, S., Pickett, L. A., and Dutt, V. Ai in healthcare: time-series forecasting using statistical, neural, and ensemble architectures. Frontiers in big data, 3:4, 2020.

Kazi, A., Cosmo, L., Navab, N., and Bronstein, M. Differentiable graph module (dgm) for graph convolutional networks. arXiv preprint arXiv:2002.04999, 2020.

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.
Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kossen, J., Band, N., Lyle, C., Gomez, A. N., Rainforth, T., and Gal, Y. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *arXiv preprint arXiv:2106.02584*, 2021.

Kurle, R., Rangapuram, S. S., de Bezenac, E., Günemann, S., and Gasthaus, J. Deep rao-blackwellised particle filters for time series forecasting. *Advances in Neural Information Processing Systems*, 33, 2020.

Lai, G., Chang, W.-C., Yang, Y., and Liu, H. Modeling long- and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 95–104, 2018.

Larson, P. D. Designing and managing the supply chain: concepts, strategies, and case studies. *Journal of Business Logistics*, 22(1):259, 2001.

Li, Y., Yu, R., Shahabi, C., and Liu, Y. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

Liberty, E., Karnin, Z., Xiang, B., Rouesnel, L., Coskun, B., Nallapati, R., Delgado, J., Sadoughi, A., Astashonok, A., Das, P., Baligol, C., Charkravarty, S., Jha, M., Gaultier, P., Januschowski, T., Flunkert, V., Wang, B., Gasthaus, J., Rangapuram, S., Salinas, D., Schelter, S., Arpin, D., and Smola, A. Elastic machine learning algorithms in Amazon SageMaker. In *Proceedings of the 2020 International Conference on Management of Data*, SIGMOD ’20, New York, NY, USA, 2020. ACM.

Lim, B. and Zohren, S. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200209, Feb 2021. ISSN 1471-2962. doi: 10.1098/rsta.2020.0209. URL http://dx.doi.org/10.1098/rsta.2020.0209.

Lim, B., Ank, S. Ö., Loeff, N., and Pfister, T. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 2021.

Montero-Manso, P. and Hyndman, R. J. Principles and algorithms for forecasting groups of time series: Locality and globality. *International Journal of Forecasting*, 2022.

Mudelsee, M. Trend analysis of climate time series: A review of methods. *Earth-science reviews*, 190:310–322, 2019.

Petropoulos, F., Apiletti, D., Assimakopoulos, V., Babai, M. Z., Barrow, D. K., Ben Taieb, S., Bergmeir, C., Bessa, R. J., Bijak, J., Boylan, J. E., and et al. Forecasting: theory and practice. *International Journal of Forecasting*, Jan 2022. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2021.11.001. URL http://dx.doi.org/10.1016/j.ijforecast.2021.11.001.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Rangapuram, S. S., Seeger, M. W., Gasthaus, J., Stella, L., Wang, Y., and Januschowski, T. Deep state space models for time series forecasting. In *Advances in Neural Information Processing Systems*, pp. 7785–7794, 2018.

Rasul, K., Sheikh, A.-S., Schuster, I., Bergmann, U., and Vollgraf, R. Multi-variate probabilistic time series forecasting via conditioned normalizing flows. *arXiv preprint arXiv:2002.06103*, 2020.

Salinas, D., Bohlke-Schneider, M., Callot, L., Medico, R., and Gasthaus, J. High-dimensional multivariate forecasting with low-rank gaussian copula processes. In *Advances in Neural Information Processing Systems 32*, 2019.

Salinas, D., Flunkert, V., Gasthaus, J., and Januschowski, T. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

Satorras, V. G., Hoogeboom, E., and Welling, M. E (n) equivariant graph neural networks. *arXiv preprint arXiv:2102.09844*, 2021.

Seo, Y., Defferrard, M., Vandergeytnst, P., and Bresson, X. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pp. 362–373. Springer, 2018.

Shang, C., Chen, J., and Bi, J. Discrete graph structure learning for forecasting multiple time series. *arXiv preprint arXiv:2101.06861*, 2021.
Shih, S.-Y., Sun, F.-K., and Lee, H.-y. Temporal pattern attention for multivariate time series forecasting. Machine Learning, 108(8):1421–1441, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In Advances in neural information processing systems, pp. 5998–6008, 2017.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. arXiv preprint arXiv:1710.10903, 2017.

Wang, Y., Smola, A., Maddix, D., Gasthaus, J., Foster, D., and Januschowski, T. Deep factors for forecasting. In International Conference on Machine Learning, pp. 6607–6617, 2019.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bromstein, M. M., and Solomon, J. M. Dynamic graph cnn for learning on point clouds. Acm Transactions On Graphics (tog), 38(5):1–12, 2019.

Wen, R., Torkkola, K., Narayanaswamy, B., and Madeka, D. A multi-horizon quantile recurrent forecaster. arXiv preprint arXiv:1711.11053, 2017.

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

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.

Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. arXiv preprint arXiv:1806.08804, 2018.

Yu, B., Yin, H., and Zhu, Z. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. arXiv preprint arXiv:1709.04875, 2017.

Zhang, R., Hao, Y., Yu, D., Chang, W.-C., Lai, G., and Yang, Y. Correlation-aware change-point detection via graph neural networks. In International Conference on Neural Information Processing, pp. 555–567. Springer, 2020.

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, 2019.
A. Model details

A.1. BP-GNN Matrices

Here, we define the adjacency matrices corresponding to the two message passing steps of the bipartite graph for $N$ time series nodes and $K$ auxiliary nodes (assuming all inference parameters $\alpha_{ij} = 1$). We also present the product of both matrices $\tilde{A} = A_2A_1$. The first $N$ rows of $\tilde{A}$ can be interpreted as the sum of all paths that communicate the $N$ time series nodes among them after applying the two message passing updates $A_2$ and $A_1$.

$$A_1 = \begin{bmatrix} 0_{N \times N} & 0_{N \times K} \\ 0_{K \times N} & 0_{K \times K} \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0_{N \times N} & 1_{N \times K} \\ 0_{K \times N} & 0_{K \times K} \end{bmatrix}, \quad \tilde{A} = A_2A_1 = \begin{bmatrix} K_{N \times N} & 0_{N \times K} \\ 0_{K \times N} & 0_{K \times K} \end{bmatrix}$$

A.2. Architecture choices $\phi_c, \phi_\alpha, \phi_h, f_{enc}$ and $f_{dec}$

In this section we will use the shortcut MLP$_{res}$ for a one layer MLP with a residual connection which we define as:

Input $\rightarrow \{\text{LinearLayer}(nf, nf) \rightarrow \text{Swish()} \rightarrow \text{LinearLayer}(nf, nf) \rightarrow \text{Addition(Input)} \} \rightarrow$ Output

Where "nf" represents the number of features.

**Decoder $f_{dec}$**

Given the above MLP$_{res}$ definition, the decoder consists of one residual MLP followed by a linear layer:

$\tilde{z}_i \rightarrow \{\text{MLP}_{res} \rightarrow \text{LinearLayer}(nf, \text{out\_dim})\} \rightarrow \tilde{x}_{i,t+1:T}$

**Encoder $f_{enc}$**

The encoder for METR-LA and PEMS-BAY consists of a linear layer and two consecutive MLP$_{res}$. Notice the overall structure can be considered an MLP with residual connections among some of its layers.

$x_{i,t};t \rightarrow \{\text{LinearLayer}(\text{in\_dim}, nf) \rightarrow \text{MLP}_{res} \rightarrow \text{MLP}_{res} \rightarrow \text{concatenate}(c_i)\} \rightarrow z_i$

In the other datasets (Solar-Energy, Traffic, Electricity and Exchange-Rate) we used the following Convolutional Neural Network as an encoder:

$x_{i,t};t \rightarrow \{\text{conv1d}(\text{in\_dim}, nf, \text{kernel\_size}, \text{stride}) \rightarrow \text{CNN}_{res}(nf) \\
\rightarrow \text{conv1d}(nf, 2nf, \text{kernel\_size}, \text{stride}) \rightarrow \text{CNN}_{res}(2nf) \\
\rightarrow \text{conv1d}(2nf, 4nf, \text{kernel\_size}, \text{stride}) \rightarrow \text{CNN}_{res}(4nf) \\
\rightarrow \text{conv1d}(4nf, \text{out\_dim}, 1, 1) \rightarrow \text{concatenate}(c_i)\} \rightarrow z_i$

Where CNN$_{res}$ follows the same architecture than MLP$_{res}$ but replacing the Linear Layers by Conv1d layers with stride=1 and kernel_size=1. In other words, both architectures are equivalent since a CNN with kernel size and stride 1 is equivalent to an MLP that broadcasts over the batch size and sequence length. Formally, we write CNN$_{res}$ as:

Input $\rightarrow \{\text{conv1d}(nf, nf/2, 1, 1) \rightarrow \text{Swish()} \rightarrow \text{conv1d}(nf/2, nf, 1, 1) \rightarrow \text{Addition(Input)} \} \rightarrow$ Output

**Edge update $\phi_c$**

Consists of two layers MLP. We divide the number of features by two in the intermediate layer for efficiency

$[b_i, h_j] \rightarrow \{\text{LinearLayer}(2\text{-nf}, nf/2) \rightarrow \text{Swish()} \rightarrow \text{LinearLayer}(nf/2, nf) \rightarrow \text{Swish()}\} \rightarrow m_{ij}$

**Node update $\phi_h$**

$[h_{i}^l, m_{ij}] \rightarrow \{\text{LinearLayer}(2\text{-nf}, nf) \rightarrow \text{Swish()} \rightarrow \text{LinearLayer}(nf, nf) \rightarrow \text{Addition}(h_{i}^l)\} \rightarrow h_{i}^{l+1}$

**Edge inference $\phi_\alpha$**

$[m_{ij}] \rightarrow \{\text{LinearLayer}(nf, 1) \rightarrow \text{Sigmoid()}\} \rightarrow \alpha_{ij}$

A.3. Linear Graph Convolutional Layer

We can define a linear graph convolutional layer $H^{l+1} = \text{GCL}(H, A)$ as:

$$Z = AH^l\theta_c + IH^l\theta_h \quad (8)$$

$$H^{l+1} = \sigma(Z) \quad (9)$$

Where $A \in \{0,1\}^{N \times N}$ is the adjacency matrix of the graph. $H \in \mathbb{R}^{N \times nf}$ a matrix of node features with $N$ nodes and "nf" features per node. $I \in \{0,1\}^{N \times N}$ is an identity matrix and $\theta_c, \theta_h \in \mathbb{R}^{nf \times nf}$ the learnable parameters.

B. METR-LA and PEMS-BAY

METR-LA is a traffic dataset collected from the highway of Los Angeles. It contains 207 nodes, a sampling resolution of 5 minutes and 34,272 samples per node (i.e. length of each time series).
PEMS-BAY is also a traffic dataset with 325 nodes located in Bay Area. The sampling resolution is also 5 minutes and it contains 52,116 samples per time series / node.

Both datasets METR-LA and PEMS-BAY can be downloaded from the following link: https://github.com/liyaguang/DCRNN

| (N) #Nodes | # Samples | Resolution | Context length | Pred. length |
|------------|-----------|------------|----------------|--------------|
| METR-LA    | 207       | 34,272     | 5 min         | 12           |
| PEMS-BAY   | 325       | 52,116     | 5 min         | 12           |

Table 8. METR-LA and PEMS-BAY specifications.

B.1. Implementation details

In this experiment (METR-LA and PEMS-BAY datasets) we used the exact same training configuration as (Shang et al., 2021). Given the whole training time series panel of dimensionality (N × Length), where Length is the number of samples of the training partition, we construct the input to the network by uniformly slicing windows $x_{t:t+12} \in \mathbb{R}^{N \times 12}$ of length 12. The ground truth labels are obtained in the same way, by slicing the next 12 time steps $x_{t+12:t+24} \in \mathbb{R}^{N \times 12}$. The same slicing process is done in validation and test with their respective partitions. The sliced windows are uniformly distributed through the whole time series panel.

All our models (FC-GNN, BP-GNN, NE-GNN) have been trained with Adam optimizer, batch size 16 and 2 layers in the GNN module. The number of features "nf" in the hidden layers is 64. The learning rates and number of epochs are provided in the following Table 9.

|        | Learning Rate | Decay at epochs | Decay factor | Epochs |
|--------|---------------|-----------------|--------------|--------|
| METR-LA| NE-GNN        | $2 \cdot 10^{-3}$ | [20, 30, 40] | 10     |
|        | FC-GNN        | $5 \cdot 10^{-3}$ | [20, 30, 40] | 10     |
|        | BP-GNN        | $2 \cdot 10^{-3}$ | [20, 30, 40] | 10     |
| PEMS-BAY| NE-GNN       | $2 \cdot 10^{-4}$ | [50]         | 10     |
|        | FC-GNN        | $2 \cdot 10^{-4}$ | [50]         | 10     |
|        | BP-GNN        | $2 \cdot 10^{-4}$ | [50]         | 10     |

Table 9. Learning rates for METR-LA and PEMS-BAY datasets

Time results have been run for batch size 16 in a Tesla V100-SXM GPU. All hyperparameters have been tuned in the validation partition using the following search spaces. The learning rate search space was $lr \in \{1 \cdot 10^{-4}, 2 \cdot 10^{-4}, 5 \cdot 10^{-4}, 1 \cdot 10^{-3}, 2 \cdot 10^{-3}, 5 \cdot 10^{-3}\}$. The number of features per layer "nf" was chosen among $nf \in \{32, 64, 128\}$. The number of layers was chosen among $\{1, 2, 3, 4\}$. The learning rate decay and the decay factor was not modified and left the same as in the original code from (Shang et al., 2021). The number of epochs was chosen large enough to do early stopping in the validation partition when the validation loss stops decreasing.

B.2. Baselines

Regarding the baselines reported in this experiment, LDS (Franceschi et al., 2019) and STGCN (Yu et al., 2017) are not evaluated on METR-LA and PEMS-BAY in their original papers, therefore, for STGCN we used the results provided in the WaveNet paper (Wu et al., 2019) which were computed by the same WaveNet authors and for LDS we used the results provided in the GTS work (Shang et al., 2021). For the DCRNN timing results we used the unofficial Pytorch implementation https://github.com/chnsh/DCRNN_PyTorch.

B.3. METR-LA and PEMS-BAY results with standard deviation

In this subsection we report the standard deviations for METR-LA and PEMS-BAY results.

C. Single step forecasting

Electricity, Solar-Energy, Electricity and Exchange Rate datasets can be downloaded from the following link: https://github.com/laiguokun/multivariate-time-series-data
All hyperparameters have been tuned in the validation partition using the following search spaces. The learning rate search space was $\mathcal{I}_{\text{lr}} \in \{5 \cdot 10^{-5}, 1 \cdot 10^{-4}, 2 \cdot 10^{-4}, 5 \cdot 10^{-4}, 1 \cdot 10^{-3}, 2 \cdot 10^{-3}\}$. The number of features $nf$ was chosen among $\{32,$
The Empirical Correlation Coefficient (CORR) is defined as:

\[ RSE = \frac{\sqrt{\sum_{(i,t) \in \Omega_{test}} (x_{it} - \hat{x}_{it})^2}}{\sqrt{\sum_{(i,t) \in \Omega_{test}} (x_{it} - \text{mean}(x))^2}} \]

The Empirical Correlation Coefficient (CORR) is defined as:

\[ CORR = \frac{1}{N} \sum_{i=1}^{N} \frac{\sum_{t} (x_{it} - \text{mean}(x_i)) (\hat{x}_{it} - \text{mean}(\hat{x})} {\sqrt{\sum_{t} (x_{it} - \text{mean}(x_i))^2 (\hat{x}_{it} - \text{mean}(\hat{x}))^2}} \]

Where \( x, \hat{x} \in \mathbb{R}^{N \times T} \).

C.3. Single step results with standard deviation

In this subsection we report the standard deviations for Solar-Energy, Traffic, Electricity and Exchange-Rate results.

| Dataset | Solar-Energy | Traffic |
|---------|--------------|---------|
| Methods | Horizon | | Horizon |
| | 3 | 6 | 12 | 24 | | 3 | 6 | 12 | 24 |
| NE-GNN | RSE | CORR | 1.898 ± 0.0018 | 2.580 ± 0.0013 | .3472 ± 0.0059 | .4441 ± 0.0083 | | 4.212 ± 0.0007 | .4586 ± 0.0017 | .4679 ± 0.0031 | .4734 ± 0.0036 | | | | |
| | RSE | CORR | 9.829 ± 0.0003 | 9.663 ± 0.0003 | .9367 ± 0.0025 | .8905 ± 0.0052 | | 8.951 ± 0.0005 | .8748 ± 0.0007 | .8700 ± 0.0015 | .8670 ± 0.0013 | | | | |
| FC-GNN | RSE | CORR | 16.51 ± 0.006 | 22.02 ± 0.002 | .2981 ± 0.0035 | .3997 ± 0.0047 | | 4.057 ± 0.0012 | .4395 ± 0.0049 | .424 ± 0.0021 | .426 ± 0.0033 | | | | |
| BP-GNN (K=4) | RSE | CORR | 9.876 ± 0.0002 | 9.765 ± 0.0003 | .9511 ± 0.0011 | .9148 ± 0.0024 | | 9.024 ± 0.0006 | .885 ± 0.0004 | .8764 ± 0.0015 | .8751 ± 0.0007 | | | | |
| | RSE | CORR | 17.04 ± 0.0017 | 22.57 ± 0.0020 | .3072 ± 0.0095 | .405 ± 0.0082 | | 4.095 ± 0.0012 | .447 ± 0.0062 | .464 ± 0.0033 | .464 ± 0.0024 | | | | |
| | RSE | CORR | 9.865 ± 0.0002 | 9.751 ± 0.0005 | .9522 ± 0.0033 | .9138 ± 0.0024 | | 8.999 ± 0.0005 | .8820 ± 0.0007 | .8744 ± 0.0015 | .8723 ± 0.0013 | | | | |

Table 14. Solar-Energy and Traffic results including standard deviations.

| Dataset | Electricity | Exchange-Rate |
|---------|--------------|---------------|
| Methods | Horizon | | Horizon |
| | 3 | 6 | 12 | 24 | | 3 | 6 | 12 | 24 |
| NE-GNN | RSE | CORR | .0762 ± .0007 | .0917 ± .0029 | .0966 ± .0018 | .0994 ± .0019 | | .0175 ± .0002 | .0244 ± .0002 | .0338 ± .0006 | .0447 ± .0008 | | | | |
| | RSE | CORR | .9494 ± .0006 | .9362 ± .0009 | .9308 ± .0006 | .9262 ± .0006 | | .9769 ± .0001 | .9686 ± .0005 | .9535 ± .0003 | .9532 ± .0003 | | | | |
| FC-GNN | RSE | CORR | 9.521 ± 0.008 | 9.404 ± 0.008 | .935 ± .0007 | .9294 ± .0006 | | 9.772 ± .0001 | .9685 ± .0004 | .9538 ± .0008 | .9349 ± .0007 | | | | |
| BP-GNN (K=4) | RSE | CORR | .0740 ± .0010 | .0898 ± .0041 | .0940 ± .0025 | .0980 ± .0019 | | .0175 ± .0001 | .0244 ± .0003 | .0339 ± .0004 | .0442 ± .0005 | | | | |
| | RSE | CORR | 9.519 ± 0.0004 | 9.396 ± 0.0007 | .9345 ± 0.0007 | .9288 ± 0.0001 | | 9.769 ± .0003 | .9684 ± .0002 | .9530 ± .0007 | .9360 ± .0011 | | | | |

Table 15. Electricity and Exchange-Rate results including standard deviations.

D. Inferred Graph Analysis

D.1. Datasets

In the Inferred graph analysis experiment we presented two synthetic datasets, "Cycle Graph" and "Correlated Sinusoids". Next, we provide a more detailed explanation on how these datasets have been generated:

- Cycle Graph: This dataset consists of a panel of N=10 time series of length T=10,000, where each time series \( i \) at time step \( t \) has been sampled from the past \( t - 5 \) of another time series \( i - 1 \mod N \) from the same panel. The resulting
multivariate time series adjacency matrix is a Cyclic Directed Graph where each variable in the panel depends on the previously indexed one. Formally the generation process is written as

\[ x_{i,t} \sim N(\beta x_{(i-1 \mod N),t-5}; \sigma^2) \]  \hspace{1cm} (10)

Where \( \beta = 0.9 \) and \( \sigma = 0.5 \).

- Correlated Sinusoids: Motivated by the Discrete Sine Transformation. This dataset consists time series of length \( T = 10,000 \), each time series generated as the weighted sum of different sinusoids with different frequencies and amplitudes. More formally we define a time series \( x_i \) as:

\[ x_i = \sum_{k=1}^{M} B_{i,m} \sin(2\pi w_{i,m} t) + \epsilon_{i,t} \]  \hspace{1cm} (11)

Where \( B_{i,k} \) is sampled from a Uniform distribution \( \tilde{B}_{i,m} \sim U(0, 1) \) and further normalized \( B_{i,m} = \frac{\tilde{B}_{i,m}}{\sum_k \tilde{B}_{i,m}} \). \( w_{i,m} \) is also sampled from a uniform distribution \( w_{i,m} \sim U(0, 0.2) \) and \( \epsilon_{i,t} \) is sampled from a Gaussian distribution \( \epsilon_{i,t} \sim N(0, 0.2^2) \). Finally, \( B_{i,m} \) and \( w_{i,m} \) are shared among different variables (i.e. time series) in the panel defining dependencies among them. We choose \( M = 3 \).

The time series in both datasets have been split into train/val/test as 6K/2K2K. In figures 5 and 6 we plot the first 200 timesteps of the training set of “Cycle Graph” and “Correlated Sinusoids” datasets respectively. For “Correlated Sinusoids” we plot the specific case of \( N = 10 \) nodes and two clusters with five nodes each \( C = \{5, 5\} \).

![Figure 5. First 200 timesteps of the Cycle Graph Gaussian dataset.](image)

**D.2. Implementation details**

In the Inferred Graph Analysis, all models have been trained for 100 epochs, learning rate of \( 2 \cdot 10^{-3} \), weight decay \( 10^{-14} \). The context length (input length of the sequence) is 6, the prediction length (output length of the sequence) is 1. The Mean
Absolute Error between prediction and ground truth has been minimized for training. We also added a small regularization value to the loss $R = 10^{-8} \sum_{i,j} \text{abs}(A_{i,j})$ that pushes unnecessary edges closer to 0 for sharper visualizations.

In both BP-GNN and FC-GNN we used the same MLP encoder than the one used METR-LA and PEMS-BAY experiments defined in the Appendix section A.2. The Graph Convolutional layer consists of only 1 layer. Since the edge inference is dynamic we obtained the reported Adjacency matrices by averaging over 10 different $t$. 

Figure 6. First 200 timesteps of Noisy Sinusoids dataset.
D.3. Inferred graph for METR-LA

In this section we analyze the adjacency matrices inferred by our FC-GNN method in METR-LA. METR-LA contains 207 nodes. As in the previous synthetic experiment, we build the FC-GNN adjacency matrices from the inferred values $A_{ij} = \alpha_{ij}$. We used the exact same training settings as in the main experiment section 5.2, but this time we only used one graph layer in the FC-GNN module from which we obtained the $\alpha_{ij}$ values. The provided adjacency has been averaged over 10 different $t$ values.

In METR-LA, sensors are spatially located around a city. Therefore, just for comparison, we also report a matrix built from the distance between each pair of sensors. We call it $A_{sim}$, where each input $A_{sim}[i, j]$ is proportional to the negative distance between the two sensors $A_{sim}[i, j] = \text{bias} - \text{dist}(i, j)$. We may expect to see some correlations between this matrix and the inferred one, but this does not have to be the case. Close sensors can be correlated, but also far away sensors can be correlated (e.g. we can expect far away residential areas to have a simultaneous increase in traffic right before working hours). Matrices are presented in the following figure, $A_{sim}$ and the inferred adjacency $A$.

![Figure 7. $A_{sim}$: matrix obtained from the negative distance among sensors. $A$: Inferred adjacency matrix by FC-GNN.](image)

Additionally, our model infers matrices dynamically for each $t$. This means that the inferred adjacency matrix can change depending on the input $x_{t_0:t}$ for different $t$ values. Following, we plot three different matrices for different $t$ values and we see that despite the overall adjacencies share similarities for different $t$, some components differ as we change $t$.

![Figure 8. Adjacency matrices for different time steps $t$.](image)