2021 IEEE 37th International Conference on Data Engineering (ICDE) | 978-1-7281-9184-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICDE51399.2021.00153

EnhanceNet: Plugin Neural Networks for Enhancing Correlated Time Series Forecasting

Razvan-Gabriel Cirstea[1], Tung Kieu[1], Chenjuan Guo[1], Bin Yang[1], Sinno Jialin Pan[2]
[1]Department of Computer Science, Aalborg University, Denmark
[2]School of Computer Science and Engineering, Nanyang Technological University, Singapore
{razvan, tungkvt, cguo, byang}@cs.aau.dk, sinnopan@ntu.edu.sg

***Abstract*—Correlated time series forecasting plays an essential role in many cyber-physical systems, where entities interact with each other over time. To enable accurate forecasting, it is essential to capture both the temporal dynamics and the correlations among different entities. To capture the former, two popular types of models, recurrent neural networks (RNNs) and temporal convolution networks (TCNs), are employed. To capture the latter, a graph is constructed to reflect certain relationships among entities and then graph convolution (GC) is applied upon the graph to capture the correlations among the entities. The state-of-the-art forecasting accuracy is achieved by models that combine RNNs or TCNs with GC. However, they neither capture distinct temporal dynamics that exist among different entities nor consider the entity correlations that evolve across time.**

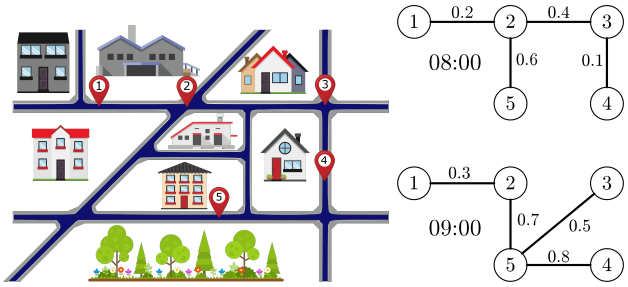
In this paper, rather than proposing yet another new end-to-end forecasting model, we aim at providing a framework to enhance existing forecasting models, where we propose generic plugins that can be easily integrated into existing solutions to solve the two challenges and thus further enhance their accuracy. Specifically, we propose two plugin neural networks that are able to better capture distinct temporal dynamics for different entities and dynamic entity correlations across time, so that forecasting accuracy is improved while model parameters to be learned are reduced. Experimental results on three real-world correlated time series data sets demonstrate that the proposed framework with the two plugin networks is able to achieve the above goals.

I. Introduction

In a complex cyber-physical system (CPS), multiple *entities* often interact with each other across time. The continued developments of sensor technologies enable recording the time-varying attributes of such entities. This produces multiple time series which correlate with each other [7]. A road transportation system is an example of such CPSs, where an entity corresponds to a road. Different sensors, e.g., loop detectors or speed cameras, are deployed in different roads to capture the speeds of the roads (see Figure 1). As traffic on different roads often affect the traffic on other roads, this produces multiple, correlated traffic speed time series.

Accurate forecasting of correlated time series plays an essential role in such CPSs, which helps reveal holistic system dynamics of the underlying CPSs, including identifying trends, predicting future behavior, and detecting outliers [17], [18]. These are important to enable effective operations of the CPSs, e.g., vehicle routing in transportation systems [12], [16], [27].

Accurate correlated time series forecasting relies on models that are able to capture two essential properties—*temporal dynamics* and *correlations* among different entities.



(a) Speed sensors. (b) Dynamic graph.

Fig. 1: Motivating example.

Temporal dynamics is important because the attributes of entities, e.g., travel speeds and traffic flows of roads, change over time and historical attributes influence future attributes [15]. For example, if a road is congested at 17:00, resulting in a low speed, the speed of the road may not be high at 17:10 as the congestion may not disappear in 10 mins. Thus, an accurate forecasting model must be able to capture such temporal dynamics by taking into account historical states when making future predictions. The state-of-the-art methods for capturing the temporal dynamics utilize deep neural networks, and can be categorized into two families: Recurrent Neural Networks (`RNNs`), e.g., Long-short Term Memory (`LSTM`) [13] and Gated Recurrent Unit (`GRU`) [6], and Temporal Convolution Networks (`TCNs`), e.g., WaveNets [24].

Entities in a CPS often interact with each other, such as traffic at different roads influencing each other. Thus, it is essential to capture the correlations among the entities to achieve accurate correlated time series forecasting. To do so, a graph is a natural choice to establish the topology of entities, where entities are modeled as vertices and relationships among entities are modeled as edges and their weights, e.g., based on Euclidean distance. Graph convolution (`GC`) [19] captures entity correlations by convoluting attributes of an entity with those of its neighbors in the graph topology, so as to obtain richer features of each entity. The entities represented with the richer features are utilized by either `RNN` or `TCN` families to enable accurate correlated time series forecasting. This kind of structures captures both temporal dynamics and entity correlations, thus yielding the state-of-the-art accuracy [21], [31], [34]. However, they fail to address two challenges.

Distinct Temporal Dynamics: Traffic at a particular road evolves across time distinctly. For example, different roads

978-1-7281-9184-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDE51399.2021.00153

may have different peak vs off-peak hours. The roads going from rural areas downtown may have more traffic in the morning, whereas the traffic on roads going from downtown to rural areas may behave oppositely. Thus, the temporal dynamics of different entities should be modeled distinctively when forecasting multiple time series.

Existing models do not distinguish temporal dynamics among different entities and treat their evolution similarly. Specifically, RNNs and TCNs used by these models apply *entity-invariant filters*, meaning that the same set of filters are used for all entities. Thus, the filters are learned to capture "average" temporal dynamics of all entities and fail to distinguish temporal dynamics among the entities. This calls for *entity-specific filters*, which apply distinct filters to different entities and hold the potential to capture distinct temporal dynamics among entities.

Dynamic Entity Correlations: Traffic on different roads influences each other and their influential levels may be different across time. An example can be seen in Figure 1, where traffic at entity 2 affects entities 1, 3 and 5 at 8 A.M., but traffic at entity 2 influences entity 5 more and no longer at entity 3 at 9 A.M. This requires a dynamic topology where the edge between entities 2 and 3 disappears from 8 A.M. to 9 A.M. In addition, it also requires dynamic edge weights where the weight between entities 2 and 5 becomes larger. Thus, a predefined and static topology employed in existing studies [21], [31] is insufficient to capture the dynamic correlations across entities. This calls for the use of a *dynamic graph*, when modeling the entity correlations.

In this paper, we aim at addressing the two challenges. Rather than proposing yet another end-to-end model, we propose a general framework *EnhanceNet* where two proposed neural network components can be easily plugged into existing models to enable them to capture distinct temporal dynamics and dynamic entity correlations, thus significantly improving time series forecasting accuracy.

We first propose a Distinct Filter Generation Network (*DFGN*) to generate distinct filters so as to capture distinct temporal dynamics of the different entities. We show how we integrate the proposed *DFGN* with both RNN and TCN family models, which not only enhance the prediction accuracy, but also effectively reduce the total number of parameters to learn.

Second, we propose a Dynamic Adjacency Matrix Generation Network (*DAMGN*) to derive dynamic graphs, with which graph convolution can capture dynamic correlations among different entities across time. We show that the proposed *DAMGN* can be easily integrated with state-of-the-art models, thus enhancing the forecasting accuracy.

To the best of our knowledge, this is the first study that proposes generic performance enhancing components for a wide variety of correlated time series forecasting models. The study makes four contributions. First, we propose a Distinct Filter Generation Network to generate distinct filters to capture distinct temporal dynamics. Second, we propose a Dynamic Adjacency Matrix Generation Network to derive dynamic adjacency matrices across time. Third, we integrate the two proposed components into popular forecasting modeling, including RNN family and TCN family. Fourth, We conduct extensive experiments using three real-world correlated time series data set to offer insight into the design choices.

Paper Outline: Section II covers related work. Section III defines the problem and presents a system overview. Sections IV and V elaborate on the two plugins for enhancing temporal dynamics and entity correlations, respectively. Sections VI shows experimental studies and Section VII concludes.

II. Related Work

We review studies on correlated time series forecasting from two aspects—the temporal dynamics and entity correlations. We also cover related work on filter generation.

Temporal dynamics: Capturing temporal dynamics is essential for time series forecasting. Various studies show that deep learning models are able to outperform non-deep learning methods [21]. Recurrent Neural Networks (RNNs) [11] and Temporal Convolution Networks (TCNs) [20] are the two most commonly used architectures for the purpose. RNNs extend feedforward neural networks by introducing hidden states to model time series step-by-step, i.e., recursively. The hidden states encodes knowledge from historical steps making RNNs is able to capture temporal dependencies among different steps. TCNs extend classic convolutional neural networks. Specifically, dilated causal convolution is applied, which only considers historical information up to the current timestamp, making it is possible to take into account temporal dependencies. WaveNet is an advanced TCN with gating mechanism, which offers good accuracy [24].

However, existing studies that use RNNs and TCNs employ the same filers for different entities, and therefore they fail to capture distinct temporal dependencies among different entities [21], [31], [34]. Naively using different filters for different entities require a very large parameter space and it does not scale w.r.t. the number of entities. To address this problem, we propose a distinct filter generation network to generate distinct filters for different entities while making the parameter space small.

Filter Generation: Filter generation has been studied for image recognition and object detection [5], [9], [32], which do not consider temporal dynamics. We consider filter generation for two model families that are able to capture temporal dynamics, i.e., RNNs and TCNs. Differently from the existing studies that the filter generator conditions on the input data, we introduce entity-specific, learnable memories to generate entity-specific filters. We offer empirical evidence to justify our design choice (see Figures 10 and 11 in Section VI).

Entity correlation: We categorize how existing studies capture entity correlations w.r.t. two dimensions. The first dimension considers if the correlations are pre-computed based on prior knowledge or it is learned from the training data. Some studies [4], [21] employ an adjacency matrix that is pre-computed using the distances among the entities to capture the correlations, and some methods employ a grid to partition the whole space and then capture the correlations among the

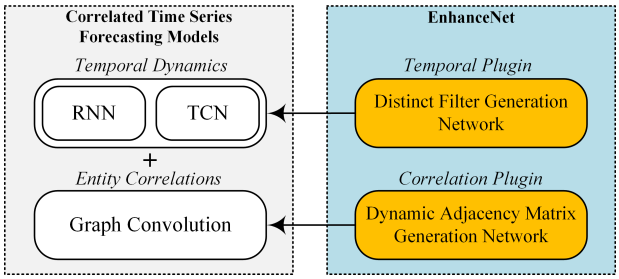


Fig. 2: Framework overview.

grid cells [29]. In contrast, `Graph WaveNet` [31] learns an adjacency matrix from the input data, which could capture correlations between two entities that are not captured by a pre-computed adjacency matrix.

The second dimension considers if the correlations may change over time, i.e., static vs. dynamic. Most existing studies consider static correlations [14], [22], [30], [31], including `Graph WaveNet` where the learned adjacency matrix is still static. Graph attention is applied to modify the weights of a given adjacency matrix, meaning that the weights are dynamic, but not the topology [8], [25]. In this paper, we fill in the gap by proposing a Dynamic Adjacency Matrix Generation Network that is able to identify dynamic typologies and also dynamic weights, which are both learned from the input data.

III. Preliminaries

A. Problem Definition

We consider a cyber-physical system with N entities, each of which is associated with a time series. The time series captures the attributes of the entity cross T timestamps, where each timestamp is associated with C attributes. Then, we use $\mathbf{X} \in \mathbb{R}^{N \times C \times T}$ to represent the time series of the N entities. Consider the example shown in Figure 1. Each entity represents a sensor, and we have $N = 5$ entities. A sensor captures $C \geq 1$ attributes of the corresponding road, e.g., speed and traffic flow, at each timestamp. We denote the time series from the i-th entity as $\mathbf{x}^{(i)} \in \mathbb{R}^{C \times T}$; we use $\mathbf{x}_t \in \mathbb{R}^{N \times C}$ to represent the time series of all the N entities at timestamp t, where each entity has C features; and we use $\mathbf{x}_t^{(i)} \in \mathbb{R}^C$ to represent the attribute vector of the i-th entity at timestamp t.

Correlated time series forecasting takes as input the past H timestamps and predicts the future F timestamps: $\mathbf{X}_H \rightarrow \mathbf{X}_F$, where $\mathbf{X}_H = [\mathbf{x}_{t-H+1}, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t] \in \mathbb{R}^{N \times C \times H}$, $\mathbf{X}_F = [\mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \ldots, \mathbf{x}_{t+F}] \in \mathbb{R}^{N \times C \times F}$, and t is current time.

B. System Overview

The framework overview is shown in Figure 2, which consists of two parts: *Correlation Time Series Forecasting Models* and *EnhnaceNet*. For the part of *Forecasting Models*, the framework supports various existing time series forecasting deep neural network models, ranging from models that explore the *temporal dynamics* of time series, such as the Recurrent Neural Network `RNN` family and the Temporal Convolution Network `TCN` family, to models that consider *entity correlations* in addition to the temporal dynamics, such as kinds of variations of *graph convolution*.

EnhnaceNet consists of a *temporal plugin* and a *correlation plugin* for enhancing *temporal dynamics* and *entity correlations*, respectively. The *temporal plugin* generates distinct filters for different entities, such that unique temporal dynamics lying in different entities can be captured distinctively. The *correlation plugin* produces dynamic adjacency among different entities across time to capture time-dependent correlations.

IV. Enhancing Temporal Dynamics

We introduce two most commonly used model families that capture temporal dynamics—Recurrent Neural Networks (`RNNs`) [11] and Temporal Convolution Neural Networks (`TCNs`) [20]. After analyzing their limitations on modeling distinct temporal dynamics for different entities, we propose a Distinct Filter Generation Network, which yields distinct filters to enable both model families to capture distinct temporal dynamics of individual entities, and thus enhance their forecasting accuracy.

A. Recurrent Neural Networks

`RNNs` are a type of neural networks that take into account historical information of a time series to enable future predictions [11]. Figure 3 shows the architecture of an `RNN` composed by a sequence of `RNN` units.

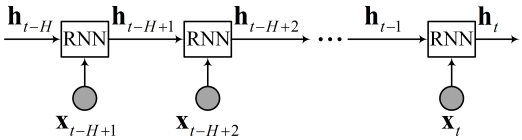


Fig. 3: A Recurrent Neural Network.

Equation 1 formalizes an `RNN` unit, where the input includes an attribute vector $\mathbf{x}_t \in \mathbb{R}^C$ observed at timestamp t and a hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^{C'}$ from the previous `RNN` unit, and the output is a new hidden state $\mathbf{h}_t \in \mathbb{R}^{C'}$ for timestamp t.

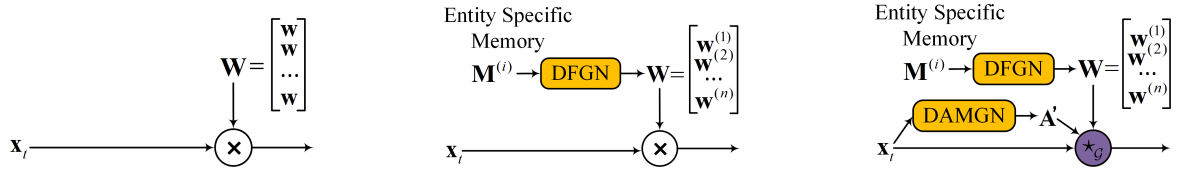
$$\mathbf{h}_t = RNN(\mathbf{x}_t, \mathbf{h}_{t-1}). \tag{1}$$

A hidden state is an abstraction that encodes useful historical knowledge of attribute vectors up to the current timestamp in the time series, and is helpful for leveraging the accuracy of future predictions. Different prediction models can be applied on top of the last hidden $\mathbf{h}_t$. The prediction model can be a fully connected neural network and can also be another `RNN`, making it an encoder-decoder architecture [21]. We then apply a standard loss function to measure the discrepancy between the predictions and the ground truth, making it possible to use back propagation to learn the model parameters.

Various implementations of `RNNs` exist. We use a simple yet effective version, Gated Recurrent Unit (`GRU`) [6], as an example to discuss the specifics. We proceed to consider a single time series from one entity and then multiple time series from different entities, resulting in `GRU` defined differently.

1) Single Time Series: We consider the time series from entity i. A `GRU` unit is given in Equation 2. Input $\mathbf{x}_t^{(i)} \in \mathbb{R}^C$ is the attribute vector at timestamp t from entity i, and $\mathbf{W} \in \mathbb{R}^{(C' \times C)}$ and $\mathbf{U} \in \mathbb{R}^{(C' \times C')}$ are the weight matrices (a.k.a., filters), to be learned through back propagation.

$$\mathbf{h}_t^{(i)} = GRU(\mathbf{x}_t^{(i)}, \mathbf{h}_{t-1}^{(i)} | \mathbf{W}, \mathbf{U}). \tag{2}$$



(a) Using same filters for all entities. (b) Using distinct filters with *DFGN*. (c) Using adjacency matrix with *DAMGN*.

Fig. 4: The fundamental operation in RNNs.

The specific computations inside a GRU unit are shown below.

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t^{(i)} + \mathbf{U}_r \mathbf{h}_{t-1}^{(i)}), \quad (3)$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_u \mathbf{x}_t^{(i)} + \mathbf{U}_u \mathbf{h}_{t-1}^{(i)}), \quad (4)$$

$$\hat{\mathbf{h}}_t = \phi(\mathbf{W}_h \mathbf{x}_t^{(i)} + \mathbf{U}_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}^{(i)})), \quad (5)$$

$$\mathbf{h}_t^{(i)} = \mathbf{u}_t \odot \mathbf{h}_{t-1}^{(i)} + (1 - \mathbf{u}_t) \odot \hat{\mathbf{h}}_t. \quad (6)$$

where $\mathbf{h}_{t-1}^{(i)}, \mathbf{h}_t^{(i)} \in \mathbb{R}^{C'}$ are hidden states, $\odot$ represents the element-wise multiplication, weight matrices $\mathbf{W}_r, \mathbf{W}_u, \mathbf{W}_h \in \mathbb{R}^{C' \times C}$, $\mathbf{U}_r, \mathbf{U}_u, \mathbf{U}_h \in \mathbb{R}^{C' \times C'}$, $\sigma(\cdot)$ is the sigmoid function, and $\phi(\cdot)$ is a non-linear activation function, e.g., $tanh$.

A GRU employs a reset gate $\mathbf{r}_t$, shown in Equation 3, to decide how much information should be forgotten from the previous timestamp. A similar gate $\mathbf{u}_t$, called update gate, is computed using Equation 4. Both the reset and update gates control how much information from the history should be considered in order to make future predictions. Specifically, the GRU computes an internal state $\hat{\mathbf{h}}_t$ that takes as input $\mathbf{x}_t^{(i)}$ and $\mathbf{h}_{t-1}^{(i)}$, and uses the reset gate $\mathbf{r}_t$, as shown in Equation 5. In Equation 6, the GRU uses the update gate $\mathbf{u}_t$ to combine the internal state $\hat{\mathbf{h}}_t$ and the hidden state $\mathbf{h}_{t-1}^{(i)}$ from the previous timestamp, and it outputs hidden state $\mathbf{h}_t^{(i)}$. By doing this, the GRU remembers historical hidden states that are relevant to future predictions and forgets those that are irrelevant.

As we could see, the fundamental operation in GRU, as well as in other RNN versions, is a matrix multiplication between an input, e.g., an attribute vector $\mathbf{x}_t^{(i)}$ or a hidden state $\mathbf{h}_t^{(i)}$, and a filter, e.g., $\mathbf{W}_r$, $\mathbf{W}_u$, $\mathbf{W}_h$, $\mathbf{U}_r$, $\mathbf{U}_u$, or $\mathbf{U}_h$.

2) Multiple Time Series: We consider the time series for all N entities, upon which the GRU is defined as:

$$\mathbf{h}_t = GRU(\mathbf{x}_t, \mathbf{h}_{t-1} | \mathbb{W}, \mathbb{U}), \quad (7)$$

where $\mathbf{x}_t \in \mathbb{R}^{N \times C}$ and $\mathbf{h}_{t-1}, \mathbf{h}_t \in \mathbb{R}^{N \times C'}$ correspond to N entities. Thus, their sizes are N times of $\mathbf{x}_t^{(i)}$ and $\mathbf{h}_{t-1}^{(i)}$ used in Equation 2. Similarly, the filters $\mathbb{W} \in \mathbb{R}^{N \times C \times C'}$ and $\mathbb{U} \in \mathbb{R}^{N \times C' \times C'}$ are also N times of filter sizes used in Equation 2.

Existing studies often apply the same filters to multiple time series from different entities. Figure 4(a) illustrates the idea for the fundamental GRU operation between an input time series $\mathbf{x}_t \in \mathbb{R}^{N \times C}$ at timestamp t and a filter $\mathbb{W} \in \mathbb{R}^{N \times C \times C'}$. Given the input time series $\mathbf{x}_t$ that covers all N entities, the filter $\mathbb{W}$ consists N copies of the same, smaller filters $W \in \mathbb{R}^{C' \times C'}$ used in Equation 2. As the filters are the same among different entities, we are unable to capture distinct temporal dynamics among different entities. The learned filters $\mathbb{W}$ represent "average" temporal dynamics over all entities. This leads to inaccurate forecasting.

Next, a straightforward way to capture distinct temporal dynamics among different entities is to use distinct filters for each entity. This means that each entity has a distinct set of filters, including three filters $\mathbf{W}_r^{(i)}$, $\mathbf{W}_u^{(i)}$, $\mathbf{W}_h^{(i)} \in \mathbb{R}^{C' \times C}$ and another three filters $\mathbf{U}_r^{(i)}$, $\mathbf{U}_u^{(i)}$, $\mathbf{U}_h^{(i)} \in \mathbb{R}^{C' \times C'}$. Thus, the filters in a GRU unit in total has $N \cdot 3 \cdot C' \cdot (C + C')$ parameters, where N is number of entities, C and C' is the size of an attribute vector and the size of a hidden state in a time series at a timestamp, respectively. C' is usually set to be large, e.g., 64, in order to make the hidden states well capture historical knowledge from time series. When N is getting large as well, the number of parameters to learn becomes overwhelming, thus resulting in the increase of model complexity and consuming large memory. The straightforward manner is often infeasible in real world settings.

This calls for a more effective method to obtain these parameters. We propose a Distinct Filter Generation Network in Section IV-C to achieve this goal.

B. Temporal Convolution Networks

A temporal convolution network (TCN) is a hierarchical model that consists of multiple layers of dilated causal convolutions to explore the temporal relationship of time series [20], [24]. Figure 5 shows the architecture of a TCN with 3 layers of dilated casual convolutions. The output of the last layer is a representation that captures temporal dynamics in the history, which enables predictions. Similar to RNN, a prediction model, e.g., a fully-connected neural network, can be used on top of the output of the last layer to make predictions. Using a loss function that measures the discrepancy between the predictions and the ground truths, standard back propagation enables learning the model parameters.

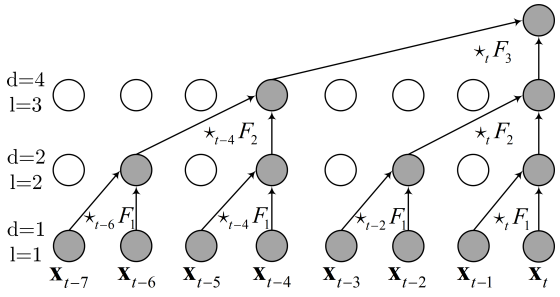


Fig. 5: Temporal Convolution Network, $K = 2$.

1) Single Time Series: We model the time series from an entity i using a TCN with l layers. To explain how a dilated causal convolution works, we first introduce two concepts—convolution filter size K and dilation factor d. Dilated causal

convolution works by convoluting K attribute vectors from K different timestamps, and attribute vectors to be convoluted are d timestamps apart. At different layers, K is often the same, while d may be different. For example, Figure 5 shows a TCN with $K = 2$, meaning that the dilated causal convolution always convolutes two attribute vectors from two timestamps. The dilation factors d in the 1st, 2nd, and 3rd layers are 1, 2, and 4, respectively. Thus, in the 1st layer, the two attributes vectors to be convoluted are 1 timestamp apart; in the 2nd layer, 2 timestamps apart; in the 3rd layer, the 4 timestamps apart, as shown in Figure 5.

We proceed to introduce the dilated causal convolution on time series $\mathbf{x}^{(\mathbf{i})}$ from entity i, as shown in Equation 8.

$$\mathbf{x}^{(\mathbf{i})} \star_t \mathbf{F}^{(i)} = \sum_{c=1}^{C} \sum_{k=1}^{K=2} \mathbf{F}^{(i)}[k, c] \odot \mathbf{x}^{(i)}_{t-d\times(k-1)}[c], \quad (8)$$

where $\star_t$ represents a convolution operation at timestamp t, $\odot$ is element-wise multiplication, and $\mathbf{F}^{(i)} \in \mathbb{R}^{K\times C}$ is the convolution filter.

Similar to traditional convolutional neural networks, where multiple filters are utilized, a TCN also uses multiple filters. In addition, the TCNs in different layers often use different sets of filters, defined in Equation 9.

$$H = TCN(\mathbf{x}^{(i)}|\mathbb{F}^{(i)}_1, \mathbb{F}^{(i)}_2, \ldots, \mathbb{F}^{(i)}_L). \quad (9)$$

where $\mathbb{F}^{(i)}_l \in \mathbb{R}^{C'\times K\times C}$ represents a set of C' dilated causal convolution filters at layer l.

2) Multiple Time Series: When modeling multiple time series with TCNs, we could use the same set of filters for all entities, but this limits the effectiveness of capturing potential distinct temporal dynamics. Otherwise, we could use a distinct set of filters for each entity, but, similar to RNNs, we can hardly afford it. For example, if we have L layers in total, we have to learn a total of $\sum_{l=1}^{L} N \cdot C'_l \cdot C_l \cdot K$ parameters in the filters, where C_l and C'_l is the size of the input vector and the number of filters at layer l, respectively. The value of C'_l can be large. This setup leads to huge memory consumption and inefficient training.

C. Distinct Filter Generation Network

To capture distinct temporal dynamics in a concise manner, we first enrich each entity with a m-dimensional memory vector $\mathbf{M}^{(i)} \in \mathbb{R}^m$, with $i \in [1, N]$. The memory is randomly initialized but trainable. Thus, the memory is able to remember entity specific temporal dynamics, and it is fed into a Distinct Filter Generation Network (*DFGN*) to generate a set of entity-specific filters $\mathbf{W}^{(i)}$ for each entity i. Specifically,

$$\mathbf{W}^{(i)} = DFGN(\mathbf{M}^{(i)}).$$

$\mathbf{W}^{(i)}$ is dependent on the underlying models, e.g., RNNs vs. TCNs. We empirically examine the learned memories in Section VI.

The *DFGN* is an abstraction of a learning model, which can be implemented in different ways. Without loss of generality, we employ a simple feed-forward neural network with two hidden layers as the *DFGN*, as shown in Figure 6. Other types of neural networks, such as convolutional neural networks, can be applied as well.

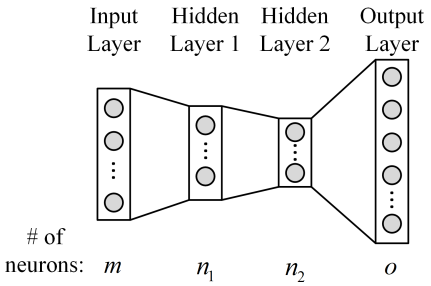


Fig. 6: Distinct Filter Generation Network (*DFGN*).

Although different entities have different memories, we only have one *DFGN*. For each entity i, we initialize a trainable memory $\mathbf{M}^{(i)}$ whose size is m and can be set as a small value. As all entities share a *DFGN*, we only need to learn a single set of parameters in *DFGN* for all entities. This significantly reduces the training complexity.

We provide an analysis on the number of parameters to be learned. Assume that each entity has a memory $\mathbf{M}^{(i)}$ with the same size m, giving rise to a total of $N \cdot m$ parameters to be learned. For the *DFGN*, the input layer consists m neurons (see Figure 6), corresponding to an entity's memory. Assuming that the two hidden layers in *DFGN* have n_1 and n_2 neurons, respectively. The output layer has o neurons, corresponding to the filters for the entity. Here, o varies when using RNN vs. TCN and we cover the specifics later. Thus, we need a total of $N \cdot m + m \cdot n_1 + n_1 \cdot n_2 + n_2 \cdot o$ parameters.

We proceed to describe how to integrate *DFGN* with RNNs and TCNs, respectively.

1) Distinct Recurrent Neural Network: Recall that the fundamental operation in an RNN unit is the multiplication between an input and a filter. Figure 4(b) illustrates the main ideas of using *DFGN* to conduct this operation. For each entity, its memory $\mathbf{M}^{(i)}$ is fed into the *DFGN*, which outputs entity-specific filters. Specifically, when using a GRU unit, the *DFGN* outputs six filters, i.e., $\mathbf{W}^{(i)}_r$, $\mathbf{W}^{(i)}_u$, $\mathbf{W}^{(i)}_h$, $\mathbf{U}^{(i)}_r$, $\mathbf{U}^{(i)}_u$, and $\mathbf{U}^{(i)}_h$, for entity i to be used in Equations 3 to 6.

Formally, we define distinct recurrent neural network (D-RNN) in Equation 10. Figure 7 shows the architecture of a D-RNN.

$$\begin{aligned} \mathbf{W}^{(i)}, \mathbf{U}^{(i)} &= DFGN(\mathbf{M}^{(i)}) \\ \mathbf{h}^{(i)}_t &= GRU(\mathbf{x}^{(i)}_t, \mathbf{h}^{(i)}_{t-1}|\mathbf{W}^{(i)}, \mathbf{U}^{(i)}). \end{aligned} \quad (10)$$

We examine the number of parameters to configure the *DFGN* for RNNs. Here, we have $o = 3C' \cdot (C + C')$ due to three filters $\mathbf{W}^{(i)}_r$, $\mathbf{W}^{(i)}_u$, $\mathbf{W}^{(i)}_h \in \mathbb{R}^{C'\times C}$ and another three filters $\mathbf{U}^{(i)}_r$, $\mathbf{U}^{(i)}_u$, $\mathbf{U}^{(i)}_h \in \mathbb{R}^{C'\times C'}$. In total, we have $N \cdot m + m \cdot n_1 + n_1 \cdot n_2 + n_2 \cdot (3 \cdot C' \cdot (C + C'))$ parameters.

We now have three methods for capturing temporal dynamics. The *naive method*, where all entities share the same

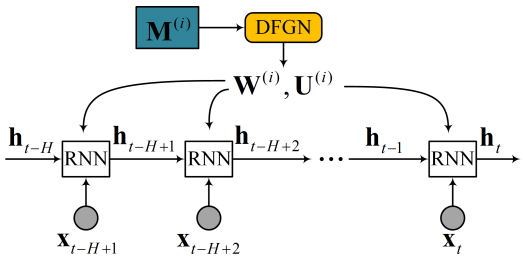


Fig. 7: Distinct Recurrent Neural Network.

sets of filters, requires learning $3 \cdot C' \cdot (C + C')$ parameters. The *straightforward method* for capturing distinct temporal dynamics requires $N \cdot (3 \cdot C' \cdot (C + C'))$ parameters, where N and C' are usually big. DRNN requires fewer parameters than the *straightforward method*, because *DFGN* is a small neural network where n_2 is often set much smaller than N. Further, DRNN could use a much smaller C' value than does the *naive method* without losing accuracy, thus requiring even fewer parameters than the *naive method*.

2) Distinct Temporal Convolution Network: In a TCN, the filters used in different dilated causal convolution layers aim to capture different granularity of temporal dynamics. Thus, if the TCN has l layers, we use l *DFGNs* to generate filters for different layers, as shown in Figure 8.

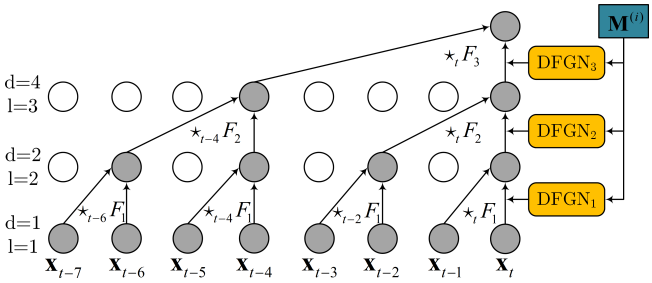


Fig. 8: Distinct Temporal Convolution Network, $K = 2$.

Formally, we define Distinct Temporal Convolution Network in Equation 11.

$$\begin{aligned} \mathbf{F}_l^{(i)} &= DFGN_l(\mathbf{M}^{(i)}) \\ H &= TCN(\mathbf{x}_t^{(i)} | \mathbb{F}_1^{(i)}, \mathbb{F}_2^{(i)}, \ldots, \mathbb{F}_l^{(i)}), \end{aligned} \tag{11}$$

where $DFGN_l$ is the *DFGN* for the l-th layer. As shown in Figure 8, for each layer, we use a unique *DFGN* to generate unique sets of filters, while the inputs to the different *DFGNs* at different layers come from the same memory vector $\mathbf{M}^{(i)}$.

Next, we investigate the total parameters required by using *DFGNs*. At each layer, the total number of parameters $o = C_l' \cdot C_l \cdot K$. Thus, in total, we have $\sum_{l=1}^{L}(m \cdot n_1 + n_1 \cdot n_2 + n_2 \cdot C_l' \cdot C_l \cdot K)$. This is significantly smaller than the *straightforward method*. In addition, except the entity memories, the number of parameters used in the *DFGNs* does not increase with the number of entities N, thus more scalable w.r.t. N.

V. Enhancing Dynamic Correlations

A. Entity Correlations

Capturing correlations among time series of different entities is essential to ensure forecasting accuracy. Existing studies model relationships among the entities as a graph, where a vertex represents an entity and an edge represents some relationship between two entities. For example, two entities are connected by an edge if their distance is smaller than a threshold. The graph is represented by an adjacency matrix.

Graph convolution (GC) [19] is applied on a graph to capture the features of an entity by aggregating and transforming those of its neighbors, so that influence upon the entity from correlated entities is taken into account. Although different variations of graph convolution exist, such as graph convolution [3], diffusion convolution [1], and spectral graph convolution [10], they all follow the same principle which can be represented in Equation 12.

$$\mathbf{Z} = \mathbf{S} \star_{\mathcal{G}} \mathbf{x}_t = \mathbf{A}\mathbf{x}_t\mathbf{S}, \tag{12}$$

where $\star_{\mathcal{G}}$ represents a GC operator, and it applies to an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, a graph signal $\mathbf{x}_t \in \mathbb{R}^{N \times C}$ denoting the C attributes of all N entities at timestamp t, and a graph convolution filter $\mathbf{S} \in \mathbb{R}^{C \times C'}$. $\mathbf{Z} \in \mathbb{R}^{N \times C'}$ is the output where each entity has C' attributes that incorporate attributes from its neighbors.

If we want to consider up to k-hop neighbors, we could replace the adjacency matrix $\mathbf{A}$ with $\mathbf{A}^k$ in Equation 12. We can also use different adjacency matrices to represent incoming neighbors and outgoing neighbors to better capture incoming and outgoing traffic flows [21].

Existing GC methods utilize a static adjacency matrix $\mathbf{A}$, and thus fail to capture entity correlations that may change over time, as shown in Figure 1. To address this limitation, we propose a Dynamic Adjacency Matrix Generation Network (*DAMGN*) to generate a unique adjacency matrix at each timestamp, so that these adjacency matrices together can capture the dynamic entity correlations across time.

We proceed to elaborate the specifics of *DAMGN*, then we describe how to integrate GC into time series forecasting models to enable accurate correlated time series forecasting.

B. Dynamic Adjacency Matrix Generation Network

The adjacency matrix derived from distances is sub-optimal because (1) it is static and fails to capture time-varying correlations; and (2) it is data-independent, and thus incapable to capture, for example, correlations between entities that are far away but still show correlations.

To address the above limitations, we utilize *DAMGN* to generate dynamic and adaptive adjacency matrices. It is dynamic since unique adjacency matrices at different timestamps are generated. It is adaptive since the adjacency matrices are derived from the input time series, which may potential capture any correlations, but not just based on, e.g., distances. Specifically, the *DAMGN* generates a new adjacency matrix

$$\mathbf{A}' = \lambda_A \mathbf{A} + \lambda_B \mathbf{B} + \lambda_C \mathbf{C}, \tag{13}$$

where $\mathbf{A}$ is the original adjacency matrix, e.g., derived from distances; $\mathbf{B}$ is a global adaptive adjacency matrix for capturing static correlations which cannot be captured by $\mathbf{A}$; and

$\mathbf{C}$ is a time-specific adaptive adjacency matrix that is derived from the time series data at a particular timestamp. Finally, the three matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are linearly combined with learnable parameters.

When using the new adjacency matrix $\mathbf{A}'$, we are able to reformulate graph convolution in Equation 14.

$$\mathbf{Z} = \mathbf{S} \star_{\mathcal{G}} \mathbf{x}_t = (\lambda_A \mathbf{A} + \lambda_B \mathbf{B} + \lambda_C \mathbf{C})\mathbf{x}_t \mathbf{S} \tag{14}$$

We proceed to cover the specifics of the *DAMGN*. The architecture of *DAMGN* is shown in Figure 9.

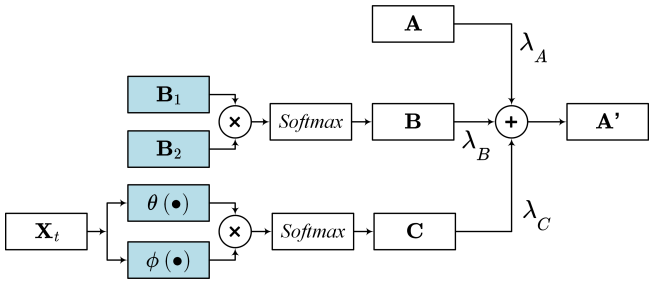


Fig. 9: Dynamic Adjacent Matrix Generation Network (*DAMGN*).

Generating B: Adaptive adjacency matrix $\mathbf{B} \in \mathbb{R}^{N \times N}$ is intended to capture hidden correlations among different entities, which cannot be captured by, e.g., distance based adjacency matrix $\mathbf{A}$.

Here, we could have initialized a learnable matrix $\mathbf{B}$ with the size of $N \cdot N$, but this brings in N^2 additional parameters, making the training unscalable, w.r.t. N, i.e., the number of entities. Instead, we follow an idea that is similar to the entity-specific memories used in the distinct filter generation network. We introduce two small memory matrices $\mathbf{B}_1, \mathbf{B}_2 \in \mathbb{R}^{N \times M}$, which could be interpreted as a source vertex memory and a target vertex memory. Note that $M \ll N$. Thus, we only need to introduce $2 \cdot N \cdot M$ additional parameters and scale well with N. Specifically, we derive adjacency matrix $\mathbf{B}$ as follows.

$$\mathbf{B} = Softmax(ReLU(\mathbf{B}_1 \mathbf{B}_2^{\mathsf{T}})) \tag{15}$$

We use $ReLU$ to remove weak correlations and $\mathbf{B}_2^{\mathsf{T}}$ indicates the transpose of matrix $\mathbf{B}_2$.

Generating C: Adjacency matrix $\mathbf{C} \in \mathbb{R}^{N \times N}$ represents a time-specific adjacency matrix, which aims at capturing correlations among entities at a specific timestamp. Specifically, $\mathbf{C}[i, j]$ captures how much entity i is affected by entity j at a specific timestamp t.

To this end, We use the normalized embedded Gaussian function to capture the correlation between $\mathbf{x}_t^{(i)}$ of entity i and $\mathbf{x}_t^{(j)}$ of entity j at timestamp t, as shown in Equation 16.

$$\mathbf{C}[i, j] = f(\mathbf{x}_t^{(i)}, \mathbf{x}_t^{(j)}) = \frac{e^{\theta(\mathbf{x}_t^{(i)})^{\mathsf{T}} \phi(\mathbf{x}_t^{(j)})}}{\sum_{j=1}^{N} e^{\theta(\mathbf{x}_t^{(i)})^{\mathsf{T}} \phi(\mathbf{x}_t^{(j)})}}. \tag{16}$$

Equation 16 resembles the attention mechanism. We first use two embedding functions θ and ϕ to perform a linear transformation on the data between entities i and j. Specifically, an embedding function multiplies an input data by a weight matrix to generate a linearly transformed data. We use two embedding functions because we want to distinguish source and target vertices, which enables us to capture asymmetric correlations between two entities.

When combining the three adjacency matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, we decide to use a weighted sum, where each adjacency matrix has its own parameter λ that controls its contribution to the final adjacency matrix. Usually, such λs are hyper-parameters but in our case instead of manually tuning them we decide to let the network learn them, thus each λ is a learnable parameter. Having such parameters also ensures that the learned adjacency matrix should be at least as powerful as the original graph convolution using only $\mathbf{A}$—when λ_B and λ_C are 0, it reduces to a normal graph convolution.

C. Correlated Time Series Forecasting

The state-of-the-art accuracy on correlated time series forecasting is achieved by integrating graph convolution (`GC`) into a model that capture temporal dynamics, e.g., `RNNs` and `TCNs`, which give rise to graph convolution `RNNs` (`GRNNs`) and graph convolution `TCNs` (`GTCNs`). We proceed to cover such integrations on the two models.

1) Integrating `GC` into `RNNs`: Integrating `GC` into `RNNs` happens inside `RNNs`. If we take a `GRU` unit as an example, the fundamental operation is the matrix multiplication between an input matrix and a weight matrix. To integrate `GC`, we replace the matrix multiplication in Equations 3, 4, 5 by the `GC` operator $\star_{\mathcal{G}}$. As an example, after the replacement, Equation 3 becomes $\mathbf{r}_t = \sigma(\mathbf{W}_r \star_{\mathcal{G}} \mathbf{x}_t + \mathbf{U}_r \star_{\mathcal{G}} \mathbf{h}_{t-1})$. Equations 4 and 5 can be updated similarly.

The replacement is reasonable because an input matrix, no matter it represents attributes in the time series or a hidden state, it can be formulated into a graph signal, where each row represents the features of an entity. After replacing matrix multiplication by `GC`, a `GRNN` is able to capture the correlations among entities when make forecasting.

Figure 4(c) illustrates the main ideas of using *DAMGN* with graph convolution in RNNs.

2) Integrating `GC` into `TCNs`: Integrating `GC` into `TCNs` happens at each layer, after the dilated causal convolution is done. Specifically, when a causal convolution finishes computations according to Equation 8, we then apply `GC` on the output of Equation 8. More specifically, the computation can be represented as: $\mathbf{x}_l = \mathbf{S} \star_{\mathcal{G}} \mathbf{x}_{l-1}$, where $\mathbf{x}_{l-1}$ represents the output of Equation 8 at the $(l-1)$-th layer, and $\mathbf{S}$ is a graph convolution filter. The computation result $\mathbf{x}_l$ is fed into the next layer, i.e., the l-th layer.

This `GC` integration is also reasonable as the output of Equation 8 can also be formulated into a graph signal, where each row represents the convoluted features of an entity.

VI. Experimental Studies

A. Experimental Setup

Data Sets: We verify the proposed *EnhnaceNet* framework for correlated time series forecasting on three public time series data sets from two different domains. In data sets *EB*

and *LA*, traffic sensors are the entities, and in data set *US*, meteorological stations are the entities.

- *EB* includes 182 traffic sensors in the East Bay area, obtained from California Transportation Agencies Performance Measurement System[1]. It covers 3 months from 1 January 2017 to 31 March 2017 and every 5 minutes we have an average speed reading from each sensor, i.e., only $C = 1$ attribute.
- *LA* includes 207 traffic sensors across Los Angeles County highways, which is released by a recent study [21]. It covers 4 months from 1 March 2012 to 30 June 2012 and every 5 minutes we get $C = 2$ attributes: an average speed and information on time and date.
- *US* includes 36 meteorological stations in the United States[2], covering 5 years from 2012 to 2017. Every hour we get $C = 6$ attributes: temperature, humidity, pressure, wind direction, wind speed, and weather description.

The three datasets are split chronologically into 3 partitions—70% for training, 10% for validation, and 20% for testing.

Forecasting setting: We consider a correlated time series forecasting problem $\mathbf{X}_H \rightarrow \mathbf{X}_F$, which is defined in Section III. We consider a commonly used setting [21], [31], where we use recent $H = 12$ timestamps as input to predict the next $F = 12$ timestamps. Following existing studies [21], [31], we report the accuracy at the 3rd, 6th, and 12th timestamps, corresponding to predictions in 15 mins, 30 mins, and 1 hour for the two traffic data sets *EB* and *LA* and predictions in 3, 6, and 12 hours for the weather data set *US*.

Graph Adjacency Matrix A: We follow existing studies to construct a graph, i.e., generating $\mathbf{A}$, using distances between entities [21], [31]. For the traffic datasets we compute the road network distances while for the weather dataset we compute the Eucledian distance of entitiy pairs to construct adjacency matrix $\mathbf{A}$ using a Gaussian kernel. The weight of the edge between sensors i and j is $\mathbf{A}_{ij} = \exp -\frac{dist(v_i,v_j)^2}{\sigma^2}$, where $dist(v_i, v_j)$ is the distance between entities i and j and σ is the standard deviation of distances. We set $\mathbf{W}_{ij} = 0$ if it is smaller than a threshold, which is set to 0.1 in the experiments.

Experiment Design: To verify our proposed plugin networks, we design two sets of experiments. First, we focus on evaluating the effects of the Distinct Filter Generation Network (*DFGN*) and Dynamic Adjacency Matrix Generation Network (*DAMGN*). More specifically, We select four base models.

- `RNN`: An encoder-decoder `RNN` model with `GRU` units [28], where the prediction model also uses a `GRU` (cf. Section IV-A).
- `TCN`: An advanced `TCN` model called `WaveNet` [24], which applies additional gating mechanism after dilated causal convolutions.
- `GRNN`: An encoder-decoder `RNN` model with modified `GRU` units with ordinary graph convolution (`GC`) [21].
- `GTCN`: Based on `WaveNet`, where ordinary `GC` is applied after the causal convolution at each layer [31].

[1]http://pems.dot.ca.gov/

[2]https://www.kaggle.com/selfishgene/historical-hourly-weather-data

For the above base models, all entities use the same set of filters, i.e., the naive method (cf. Section IV). Following existing studies [21], [31], the ordinary `GC` in `GRNN` and `GTCN` employ two distance-derived, static adjacency matrices, representing both incoming and outgoing neighbors. In addition, we consider neighbors up to 2 hops.

Since `RNN` and `TCN` only captures temporal dynamics, we evaluate how *DFGN* is able to improve `RNN` and `TCN`. More specifically, we consider two models:

- `D-RNN`: `RNN` with *DFGN*.
- `D-TCN`: `TCN` with *DFGN*.

Next, `GRNN` and `GTCN` capture both temporal dynamics and entity correlations, we evaluate how both *DFGN* and *DAMGN* may enhance accuracy. More specifically, we consider six additional models.

- `D-GRNN`: `GRNN` with only *DFGN*.
- `DA-GRNN`: `GRNN` with only *DAMGN*.
- `D-DA-GRNN`: `GRNN` with both *DFGN* and *DAMGN*.
- `D-GTCN`: `GTCN` with only *DFGN*.
- `DA-GTCN`: `GTCN` with only *DAMGN*.
- `D-DA-GTCN`: `GTCN` with both *DFGN* and *DAMGN*.

We regard the first set of experiments as an ablation study.

Second, we compare the final models, i.e., `D-DA-GRNN` and `D-DA-GTCN` with multiple baselines including the state-of-the-art.

- `ARIMA` [2]: Auto-Regressive Integrated Moving Average model with Kalman filter. A non-deep learning baseline.
- `LSTM` [13]: Another popular `RNN` structure with long-short term memory (`LSTM`) units. Like `GRU`, an encoder-decoder architecture is used to make predictions.
- `WaveNet` [24]: The same with `TCN`.
- `DCRNN` [21]: Diffusion convolution recurrent neural network. It is the state-of-the-art model for correlated time series forecasting in the `RNN` family.
- `STGCN` [34]: Spatial-temporal graph convolution network that combines 1D convolution with `GC` in a non-hierachical way.
- `Graph WaveNet` [31]: The state-of-the-art model for correlated time series forecasting in the `TCN` family. It also achieves the best accuracy so far.

We do not include `ST-MetaNet` [25] because `ST-MetaNet` employs additional prior knowledge such as POIs, whereas all the above methods do not. In addition, the accuracy of `ST-MetaNet` is worse than `Graph WaveNet`.

Evaluation Metrics: We consider both accuracy and model complexity. For accuracy, we follow existing literature to report mean absolute error (*MAE*), root mean square error (*RMSE*), and mean absolute percentage error (*MAPE*) [21], [31]. For model complexity, we report the total number of parameters to be learned for each model.

Model Configurations: For `RNN` based models, we use encoder-decoder architectures, with 2 `GRU` layers stacked on each other. All `GRU` units use $C' = 64$. We train all models using an initial learning rate of 0.01 which reduces $\frac{1}{10}$ every 10 epochs starting with the 20th epoch for a maximum of 100

iterations. In addition, scheduled sampling is used. Existing studies suggest the above setting is effective [8], [21].

For TCN based models, we stack $L = 8$ layers, where the layers have dilation factors of 1, 2, 1, 2, 1, 2, 1, and 2. We set the causal convolution filter size $K = 2$, with a total of $C' = 32$ filters. We train the models using a fixed learning rate of 0.001 and a dropout of 0.3. We use this setting since an existing study suggests this setting is effective [31].

For *DFGN*, we use a 2-layer fully connected network, with $n_1 = 16$ and $n_2 = 4$ neurons. We randomly initialize each entity's memory with size $m = 16$ using a uniform distribution. For *DAMGN*, we set $M = 10$ to initialize the memory matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ (cf. Section V-B).

Implementation Details: We implement all models on Python 3.7.3 using PyTorch 1.3.1. The experiments are conducted on a computer node on the CLAAUDIA cloud (www.claaudia.aau.dk), running Ubuntu 16.04.6 LTS, with one Intel(R) Xeon(R) CPU @2.50GHz and one Tesla V100 GPU card. The source code of all models will be made public available upon acceptance.

B. Experimental Results

1) Effectiveness of DFGN: To justify our design choice on having distinct filters for different entities, we design experiments in which we compare a modle without the *DFGN* vs. the same model with *DFGN*.

Accuracy enhancement on RNN and TCN: Table I reports the results on RNN and TCN. The results are first grouped per data set. For each data set, the results are further grouped into RNN vs. TCN based models. For each data set, we use bold to highlight the lowest errors, suggesting the best prediction accuracy. Within each model family, we use underline to show the lowest error among the same family.

We consider the first data set *EB*. We can see that the error values for D-RNN are always with underlines, suggesting that they are smaller than those of RNN. Thus, D-RNN improves accuracy over RNN. For TCN family, we observe the same pattern—D-TCN outperforms TCN in all settings. Finally, the bold values are shown either D-RNN and D-TCN, meaning that the *DFGN* enhanced models achieve the best results.

For the other two data sets *LA* and *US*, we see similar trends. Thus, we conclude that *DFGN* is able to enhance the forecasting accuracy for both RNNs and TCNs.

Model Complexity: We report the number of parameters for each model in the "# Para" column in Table I. The model with *DFGN* requires significantly less parameters. This is because we can use smaller generated filters while maintaining high accuracy. For example, for RNN we need to use $C' = 64$. In contrast, for D-RNN, we use $C' = 16$, which is already more accurate than RNN.

Memories Learned from *DFGN*: Next, we investigate what *DFGN* eventually learns. To do this, we examine the learned memory for each entity from the *DFGN*. We apply t-SNE [23] to compress each entity's memory learned by D-TCN from $m = 16$ to 2 dimensions, shown in Figure 10. The memories of different entities are spread over the 2D space. This indicates that each entity has a distinct memory, and thus the DFGN is able to generate distinct filters to capture various temporal patterns for different entities. Using the same filters for all entities fail to capture such distinct temporal dynamics, which give less accurate results as shown in Table I.

Next, we investigate nearby memories in Figure 10. We use four different colors to highlight 4 clusters of nearby memories in Figure 10. We map the locations of the corresponding entities, i.e., sensors, on a map shown in Figure 11 using the same color. We observe that each memory cluster represents a group of sensors that are along a short segment of a highway, which are supposed to have similar temporal dynamics.

In addition, we observe that the red and black sensors in Figure 11 are nearby. However their memories captured by *DFGN* are further apart (see Figure 10), meaning that although the sensors are close to each other, they exhibit different temporal patterns, e.g., due to that the black ones are on a north-south road while the red ones are one a west-east road.

We conclude that *DFGN* is effective because it is able to (1) capture distinct temporal dynamics for different sensors; (2) capture similar temporal dynamics of sensors that are along a short segment of a road; and (3) distinguish sensors that are nearby but have different temporal patterns.

We next study the sensitivity of the size of memories m in Table IV. Using larger memories achieve only slightly smaller average error values. Thus, parameter m is insensitive to the final accuracy, making it easy to configure *DFGN*.

Accuracy enhancement on GRNN and GTCN: Table 12 reports the results on GRNN and GTCN. We observe that D-GRNN and D-GTCN often get lower error values than GRNN and GTCN do, especially for long term predictions where temporal dynamics matters more.

2) Effectiveness of DAMGN: To verify the effectiveness of the learned dynamic adjacency matrices, we investigate the GRNN and GTCN models with/without using *DAMGN*.

Accuracy enhancement: The results are shown in Table II. We also group results per data set first and then per model family. Bold values suggest the lowest errors per data set and underline values indicate the lowest errors per model family.

We can see that in all three data sets, DA-GRNN has smaller errors than GRNN and DA-GTCN has smaller errors than GTCN. This suggests that using dynamic adjacency matrix helps the models better capture dynamic entity correlations, which eventually improve prediction accuracy.

Next, we observe that when combining both *DFGN* and *DAMGN*, it often achieves the best accuracy, especially for the GRNN family.

Model complexity: The models with *DAMGN* have slightly more parameters than the base models, as it introduces additional parameters to computer adjacency matrices $\mathbf{B}$ and $\mathbf{C}$. When using both *DFGN* and *DAMGN* together, the additional parameters introduced by *DAMGN* becomes negligible. Since *DFGN* reduces significantly the number of parameters, the "D-DA-" models have much less parameters than the base models.

Data	Models	3^{rd} *timestamp*			6^{th} *timestamp*			12^{th} *timestamp*			# Para
		MAE	*MAPE*	*RMSE*	*MAE*	*MAPE*	*RMSE*	*MAE*	*MAPE*	*RMSE*	
EB	RNN	3.34	6.67	6.32	4.03	8.48	7.90	5.17	11.41	10.07	74k
	D-RNN	**3.28**	6.52	**6.18**	3.90	8.20	**7.62**	4.83	10.76	**9.44**	49k
	TCN	3.37	6.65	8.08	4.05	8.51	9.32	5.12	11.43	11.10	247k
	D-TCN	3.30	**6.45**	7.78	**3.89**	**8.05**	8.91	**4.76**	**10.41**	10.43	100k
LA	RNN	3.03	8.09	5.98	3.69	10.46	7.48	4.69	14.13	9.35	75k
	D-RNN	2.89	7.79	5.73	3.36	9.70	6.94	3.90	12.03	8.24	49k
	TCN	2.98	7.91	5.89	3.58	10.18	7.27	4.44	13.61	8.95	247k
	D-TCN	**2.85**	**7.59**	**5.67**	**3.29**	**9.37**	**6.82**	**3.79**	**11.41**	**8.02**	103k
US	RNN	1.25	0.43	1.83	1.91	0.66	2.65	2.11	0.73	2.89	75k
	D-RNN	1.25	0.43	1.83	1.90	0.65	2.65	2.14	0.74	2.95	32k
	TCN	1.15	0.40	1.67	1.78	0.62	2.47	2.07	0.72	2.84	247k
	D-TCN	**1.12**	**0.39**	**1.63**	**1.70**	**0.59**	**2.37**	**1.99**	**0.69**	**2.75**	104k

TABLE I: Effect of *DFGN* on capturing distinct temporal dynamics.

Data	Models	3^{rd} *timestamp*			6^{th} *timestamp*			12^{th} *timestamp*			# Para
		MAE	*MAPE*	*RMSE*	*MAE*	*MAPE*	*RMSE*	*MAE*	*MAPE*	*RMSE*	
EB	GRNN	3.20	6.41	5.96	3.68	7.75	7.11	4.35	9.42	8.52	371k
	D-GRNN	3.17	6.34	5.88	3.59	7.57	6.94	4.15	8.96	8.15	167k
	DA-GRNN	3.07	6.10	5.70	3.46	7.14	6.71	3.95	8.25	7.77	378k
	D-DA-GRNN	**3.05**	6.09	**5.67**	**3.40**	**7.12**	**6.57**	**3.83**	**8.17**	**7.53**	172k
	GTCN	3.15	6.18	7.46	3.60	7.44	8.33	4.21	8.86	9.39	288k
	D-GTCN	3.17	6.22	7.47	3.58	7.37	8.25	4.15	8.77	9.24	252k
	DA-GTCN	3.10	**6.07**	7.38	3.50	7.16	8.13	3.96	8.35	8.96	296k
	D-DA-GTCN	3.11	6.14	7.43	3.45	7.15	8.09	3.87	8.09	8.82	195k
LA	GRNN	2.77	7.30	5.38	3.15	8.80	6.45	3.60	10.50	7.60	372k
	D-GRNN	2.66	6.81	5.11	3.04	8.24	6.17	3.47	9.93	7.32	172k
	DA-GRNN	2.67	6.86	5.17	3.06	8.27	6.25	3.49	9.86	7.38	381k
	D-DA-GRNN	**2.64**	**6.75**	**5.07**	**3.02**	**8.19**	**6.12**	**3.45**	**9.85**	7.27	180k
	GTCN	2.72	6.94	5.21	3.12	8.46	6.27	3.59	10.20	7.43	288k
	D-GTCN	2.71	6.85	5.23	3.08	8.28	6.27	3.55	9.96	7.43	113k
	DA-GTCN	2.71	7.00	5.12	3.08	8.44	6.20	3.53	10.06	7.25	297k
	D-DA-GTCN	2.69	6.93	5.14	3.06	8.29	6.19	3.49	9.96	**7.23**	261k
US	GRNN	0.89	**0.30**	1.32	1.28	0.44	1.84	1.56	0.54	2.22	226k
	D-GRNN	0.92	0.31	1.36	1.30	0.45	1.89	1.57	0.54	2.23	96k
	DA-GRNN	0.88	**0.30**	1.30	1.24	**0.42**	1.78	1.52	**0.52**	2.15	227k
	D-DA-GRNN	0.88	**0.30**	1.29	1.23	**0.42**	1.75	**1.50**	**0.52**	**2.13**	97k
	GTCN	0.90	0.31	1.33	1.30	0.45	1.86	1.58	0.55	2.24	272k
	D-GTCN	0.91	0.31	1.35	1.29	0.45	1.85	1.59	0.55	2.24	111k
	DA-GTCN	0.88	**0.30**	1.30	1.26	0.44	1.80	1.55	0.54	2.20	273k
	D-DA-GTCN	**0.86**	**0.30**	**1.27**	**1.22**	**0.42**	**1.74**	1.53	0.53	2.15	156k

TABLE II: Effect of *DFGN* and *DAMGN* on capturing distinct temporal dynamics and entity correlations.

Data	Models	3^{rd} *timestamp*			6^{th} *timestamp*			12^{th} *timestamp*		
		MAE	*MAPE*	*RMSE*	*MAE*	*MAPE*	*RMSE*	*MAE*	*MAPE*	*RMSE*
EB	ARIMA	6.08	11.65	10.75	6.48	13.22	11.55	6.61	13.81	10.87
	FC-LSTM	4.17	8.67	7.83	4.31	9.08	8.16	4.56	9.90	8.70
	WaveNet	3.37	6.65	8.08	4.05	8.51	9.32	5.12	11.43	11.10
	STGCN	4.37	8.45	10.83	4.81	9.67	11.48	5.56	11.82	12.53
	Graph WaveNet	3.10	6.05	7.36	3.47	7.07	8.07	3.93	8.18	8.86
	DCRNN	3.20	6.41	5.96	3.68	7.75	7.11	4.35	9.42	8.52
	D-DA-GTCN	3.11	6.14	7.43	3.45	7.15	8.09	3.87	**8.09**	8.82
	D-DA-GRNN	**3.05**	**6.04**	**5.67**	**3.40**	**7.05**	**6.57**	**3.83**	8.17	**7.53**
LA	ARIMA	3.99	9.60	8.21	5.15	12.70	10.45	6.90	17.40	13.23
	FC-LSTM	3.44	9.60	6.30	3.77	10.90	7.23	4.37	13.20	8.69
	WaveNet	2.99	8.04	5.89	3.59	10.25	7.28	4.45	13.62	8.93
	STGCN	2.88	7.62	5.74	3.47	9.57	7.24	4.59	12.70	9.40
	Graph WaveNet	2.69	6.90	5.15	3.07	8.37	6.22	3.53	10.01	7.37
	DCRNN	2.77	7.30	5.38	3.15	8.80	6.45	3.60	10.50	7.59
	D-DA-GTCN	2.69	6.93	5.14	3.06	8.29	6.19	3.49	9.96	**7.23**
	D-DA-GRNN	**2.64**	**6.75**	**5.07**	**3.02**	**8.19**	**6.12**	**3.45**	**9.85**	**7.23**
US	ARIMA	2.02	0.73	2.69	3.78	1.37	4.54	5.61	2.05	6.95
	FC-LSTM	1.68	0.58	2.30	1.84	0.63	2.54	2.02	0.70	2.77
	WaveNet	1.15	0.40	1.67	1.78	0.62	2.47	2.07	0.72	2.84
	STGCN	1.74	0.60	2.31	1.94	0.68	2.58	2.03	0.71	2.69
	Graph WaveNet	0.87	0.31	1.28	1.23	0.43	1.76	1.54	0.55	2.19
	DCRNN	0.89	**0.30**	1.32	1.28	0.44	1.84	1.56	0.54	2.22
	D-DA-GTCN	**0.86**	**0.30**	**1.27**	**1.22**	**0.42**	**1.74**	1.53	0.53	2.15
	D-DA-GRNN	**0.88**	**0.30**	1.29	1.23	**0.42**	1.75	**1.50**	**0.52**	**2.13**

TABLE III: Comparison with baselines and state-of-the-art methods.

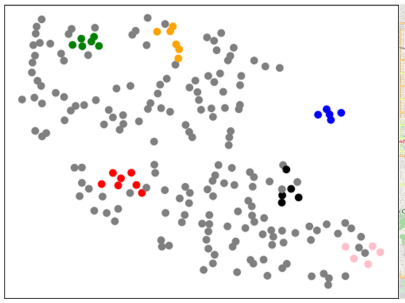


Fig. 10: Entity memories, D-TCN, *LA*.

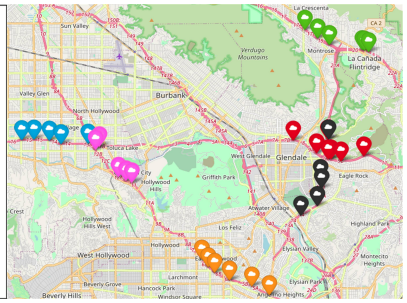


Fig. 11: Entity locations, *LA*.

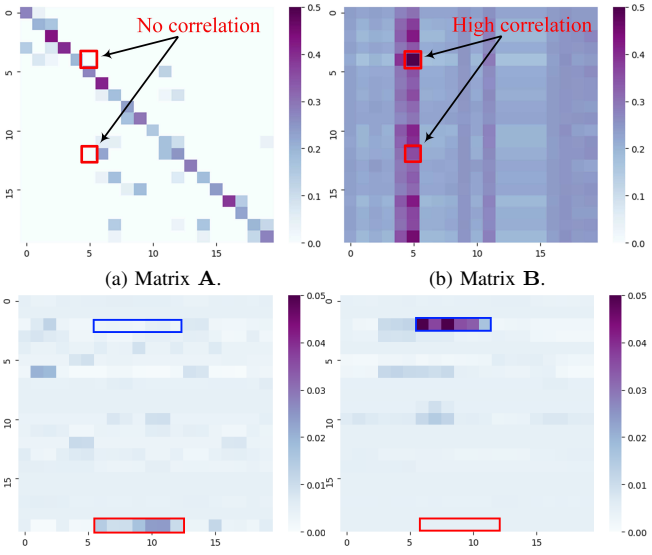
m	*MAE*	*MAPE*	*RMSE*
8	4.05	10.19	8.28
16	4.04	10.15	8.22
18	4.04	10.11	8.20
32	4.03	10.10	8.18

TABLE IV: Sensitivity of m, D-TCN.

Adjacency Matrices Learned from *DAMGN*: We take a closer look at the learned adjacency matrices $\mathbf{B}$ and $\mathbf{C}$. Specifically, we select 20 sensors and examine their corresponding matrices $\mathbf{B}$ and $\mathbf{C}$ learned from DA-TCN in Figure 12.

First, we compare $\mathbf{A}$ and $\mathbf{B}$. Recall that $\mathbf{A}$ is derived based on distances and is static. The learned matrix $\mathbf{B}$ is also static, but is learned from additional memory matrices $\mathbf{B}_2$ and $\mathbf{B}_2$ (cf. Section V-B). The two matrices are different—see the highlighted cells. This indicates that distance based adjacency matrix may not fully capture entity correlations.

Second, we compare the $\mathbf{C}$ matrix at two different timestamps since it is dynamic. We observe that the two matrices are very different. In the first timestamp, sensor 19 has some correlations with sensors 6 to 12 (see the red rectangles), while in the second timestamp, sensor 19 is not correlated with them anymore. In the second timestamp, sensor 2 is heavily correlated with sensors 5 to 10 (see the blue rectangles), while in the first timestamp, there are almost no correlations. This confirms that static adjacency matrix is insufficient as entity correlations are dynamic.



(a) Matrix $\mathbf{A}$. (b) Matrix $\mathbf{B}$.

(c) Matrix $\mathbf{C}$ at two different timestamps.

Fig. 12: Learned adjacency matrices, DA-TCN, *LA*.

3) Comparisons with Baselines: We compare the best two models D-DA-GRNN and D-DA-GTCN from our proposals with multiple methods, including the state-of-the-art of the RNN and TCN families—DCRNN and Graph WaveNet.

The results are shown in Table III. We observe that all deep learning methods performs better then non-deep learning baselines such as ARIMA, since the temporal dynamics are highly non-linear. In addition we can see that combining *DFGN* and *DAMGN* offers a significant accuracy improvement, since either D-DA-GTCN or D-DA-GRNN achieves the best accuracy on all three data sets. We also observe that D-DA-GRNN performs better than D-DA-GTCN. D-DA-GRNN outperforms the state-of-the-art method Graph WaveNet, while D-DA-GTCN sometimes falls behind Graph WaveNet. Next, we perform the t-tests to test the significance of the proposed models D-DA-GRNN and D-DA-GTCN compared to the state-of-the-art methods DCRNN and Graph WaveNet. The p-values are less than 0.01 for all three datasets when using all three error metrics. This provides a strong evidence that the proposed models statistically outperforms the state-of-the-art methods.

4) Runtime: We study runtime for both offline training and online predictions in Table V. For training, we report average runtime (seconds) per epoch in the "T (s)" column. For predictions, we report the average runtime (milliseconds) for making a prediction for the next 12 timestamps in the "P (ms)" column.

Training runtime: Using *DFGN* increases the runtime. This is not surprising as each training iteration has to go through the additional *DFGN* for generating the entity specific filters. The run-time increase on D-TCN is larger than that on D-RNN because D-TCN maintains multiple *DFGNs*, one for each dilation convolution layer whereas D-RNN maintains only one.

Using *DAMGN* only increase slightly the runtime, due to its light design in Equations 15 and 16, with only a few more matrix multiplications.

Prediction runtime: In the prediction phase, we do not need to use *DFGN* anymore as the dynamic filters are already identified in the training phase. Thus, the runtime of the "D-" models is very similar to their base models. Although *DAMGN* is still needed to generate dynamic adjacency matrix $\mathbf{C}$ based on the input time series, the computation in *DAMGN* (cf. Equation 16) is light. Thus, the runtime of the "DA-" models is also similar to their base models. Thus, we conclude that the use of *DFGN* and *DAMGN* does not affect the usability in real-time predictions.

Models	EB		LA		US	
	T (s)	P (ms)	T (s)	P (ms)	T (s)	P (ms)
RNN	17.4	0.31	25.5	0.35	25.9	0.29
D-RNN	24.8	0.31	36.1	0.33	34.6	0.29
TCN	21.8	0.33	33.6	0.16	14.4	0.28
D-TCN	76.7	0.23	108.7	0.27	46.1	0.23
GRNN	54.4	1.31	88.1	1.59	50.2	0.60
D-GRNN	74.4	0.95	115.7	1.18	59.1	0.57
DA-GRNN	66.0	1.35	106.9	1.51	50.8	0.60
D-DA-GRNN	82.6	0.90	129.5	1.10	60.5	0.52
GTCN	29.8	0.25	45.9	0.29	17.6	0,18
D-GTCN	77.7	0.27	131.3	0.33	50.3	0,19
DA-GTCN	34.7	0.27	53.6	0.32	19.6	0.19
D-DA-GTCN	85.0	0.29	129.6	0.32	51.7	0.20

TABLE V: Runtime.

5) Summary: Based on both the accuracy and runtime studies, we make the following recommendations. First, if training time is critical, we should only use *DAMGN*, which has a similar training runtime with the base models and also provides good accuracy enhancement. Second, if training time is not critical, we recommend use both *DFGN* and *DAMGN*, which gives the largest accuracy enhancement.

VII. Conclusion and Outlook

We present a framework with two plugin networks—Distinct Filter Generation Network *DFGN* and Dynamic Adjacency Matrix Generation Network *DAMGN*, which can be employed to enhance the accuracy of many time series forecasting models. This is due to the two plugin networks are able to capture distinct temporal dynamics among entities and dynamic entity correlations, which existing models fail to capture. In addition, the two plugin networks are also able to reduce the total number of parameters. In future work, it is of interest to integrate with CPSs such as intelligent transportation systems [26], [33].

Acknowledgements

This work was supported by Independent Research Fund Denmark under agreements 8022-00246B and 8048-00038B.

References

[1] Atwood, J., and Towsley, D. Diffusion-convolutional neural networks. *NIPS* (2016), 1993–2001.

[2] Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[3] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *CoRR abs/1312.6203* (2013).

[4] Chen, C., Li, K., Teo, S. G., Zou, X., Wang, K., Wang, J., and Zeng, Z. Gated residual recurrent graph neural networks for traffic prediction. In *AAAI* (2019), pp. 485–492.

[5] Chen, Y., Dai, X., Liu, M., Chen, D., Yuan, L., and Liu, Z. Dynamic convolution: Attention over convolution kernels. In *CVPR* (2020), pp. 11030–11039.

[6] Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR abs/1412.3555* (2014).

[7] Cirstea, R., Micu, D., Muresan, G., Guo, C., and Yang, B. Correlated time series forecasting using multi-task deep neural networks. In *CIKM* (2018), pp. 1527–1530.

[8] Cirstea, R.-G., Guo, C., and Yang, B. Graph attention recurrent neural networks for correlated time series forecasting. In *MileTS@KDD19* (2019).

[9] De Brabandere, B., Jia, X., Tuytelaars, T., and Van Gool, L. Dynamic filter networks. In *NIPS* (2016), pp. 667—-675.

[10] Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 2016, pp. 3844–3852.

[11] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[12] Guo, C., Yang, B., Hu, J., Jensen, C. S., and Chen, L. Context-aware, preference-based vehicle routing. *VLDB J. 29*, 5 (2020), 1149–1170.

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

[14] Hu, J., Guo, C., Yang, B., and Jensen, C. S. Stochastic weight completion for road networks using graph convolutional networks. In *ICDE* (2019), pp. 1274–1285.

[15] Hu, J., Yang, B., Guo, C., Jensen, C. S., and Xiong, H. Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks. In *ICDE* (2020), pp. 1417–1428.

[16] Kieu, T., Yang, B., Guo, C., and Jensen, C. S. Distinguishing trajectories from different drivers using incompletely labeled trajectories. In *CIKM* (2018), pp. 863–872.

[17] Kieu, T., Yang, B., Guo, C., and Jensen, C. S. Outlier detection for time series with recurrent autoencoder ensembles. In *IJCAI* (2019), pp. 2725–2732.

[18] Kieu, T., Yang, B., and Jensen, C. S. Outlier detection for multidimensional time series using deep neural networks. In *MDM* (2018), pp. 125–134.

[19] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. (Poster).

[20] Lea, C., Flynn, M. D., Vidal, R., Reiter, A., and Hager, G. D. Temporal convolutional networks for action segmentation and detection. In *CVPR* (2017), pp. 156–165.

[21] Li, Y., Yu, R., Shahabi, C., and Liu, Y. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *ICLR* (2018), pp. 1–16.

[22] Liang, Y., Ke, S., Zhang, J., Yi, X., and Zheng, Y. Geoman: Multi-level attention networks for geo-sensory time series prediction. In *IJCAI* (2018), pp. 3428–3434.

[23] Maaten, L. v. d., and Hinton, G. Visualizing data using t-sne. *JMLR 9*, Nov (2008), 2579–2605.

[24] Oord, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499* (2016).

[25] Pan, Z., Liang, Y., Wang, W., Yu, Y., Zheng, Y., and Zhang, J. Urban traffic prediction from spatio-temporal data using deep meta learning. In *SIGKDD* (2019), pp. 1720–1730.

[26] Pedersen, S. A., Yang, B., and Jensen, C. S. Anytime stochastic routing with hybrid learning. *Proc. VLDB Endow. 13*, 9 (2020), 1555–1567.

[27] Pedersen, S. A., Yang, B., and Jensen, C. S. Fast stochastic routing under time-varying uncertainty. *VLDB J. 29*, 4 (2020), 819–839.

[28] Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *NIPS* (2014), Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., pp. 3104–3112.

[29] Wang, D., Cao, W., Li, J., and Ye, J. Deepsd: Supply-demand prediction for online car-hailing services using deep neural networks. In *ICDE* (2017), pp. 243–254.

[30] Wang, J., Gu, Q., Wu, J., Liu, G., and Xiong, Z. Traffic speed prediction and congestion source exploration: A deep learning method. In *ICDM* (2016), pp. 499–508.

[31] Wu, Z., Pan, S., Long, G., Jiang, J., and Zhang, C. Graph wavenet for deep spatial-temporal graph modeling. In *IJCAI* (2019), pp. 1907–1913.

[32] Yang, B., Bender, G., Le, Q. V., and Ngiam, J. Condconv: Conditionally parameterized convolutions for efficient inference. In *NIPS* (2019), pp. 1307–1318.

[33] Yang, S. B., Guo, C., and Yang, B. Context-aware path ranking in road networks. *IEEE Trans. Knowl. Data Eng.* (2020).

[34] Yu, B., Yin, H., and Zhu, Z. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *IJCAI* (2018), pp. 3634–3640.