# Time-Series Forecasting using Dynamic Graphs: Case Studies with Dyn-STGCN and Dyn-GWN on Finance and Traffic Datasets

Shibal Ibrahim*
shibal@mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

Max R. Tell*
maxtell@mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

Rahul Mazumder
rahulmaz@mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

## ABSTRACT

Spatio-temporal modeling is an essential lens to understand many real-world phenomena from traffic to epidemiology. Although forecasting time-series is an exceptionally well-studied problem, recent years have seen impressive gains in the performance of graph learning as a paradigm for spatial learning problems. Some recent work has explored the intersection of these two fields, but often assumes that the spatial structure is static. We propose a new framework for spatio-temporal learning from dynamic graphs. The two main components of our models are: (i) Temporal convolutions on the time-varying adjacency space, (ii) Tensor Graph Convolutional Layer (TGCL) which aggregates latent temporal representations of time-varying node features and time-varying graphs. We generalize previous models to leverage both dynamic and static graphs while being attractive in terms of computational efficiency. We demonstrate our proposals with two new time-varying graph-based methods Dyn-STGCN and Dyn-GWN for time-series forecasting. Experiments demonstrate the efficacy of these model across datasets from different domains. Interestingly, our Dyn-STGCN and Dyn-GWN models are superior at handling dynamic graphs than existing state-of-the-art time-varying graph-based methods e.g., EvolveGCN and TM-GCN in terms of strong generalization while providing both efficient training and inference.

## CCS CONCEPTS

• **Information systems → Spatial-temporal systems**; • **Computing methodologies → Neural networks**; **Temporal reasoning**; • **Mathematics of computing → Time series analysis**; • **Applied computing → Transportation**; **Forecasting**.

## KEYWORDS

spatio-temporal modeling; time-series forecasting; graph neural networks; dynamic graphs

*Both authors contributed equally to this research.

**ACM Reference Format:**
Shibal Ibrahim, Max R. Tell, and Rahul Mazumder. 2022. Time-Series Forecasting using Dynamic Graphs: Case Studies with Dyn-STGCN and Dyn-GWN on Finance and Traffic Datasets. In *8TH SIGKDD International Workshop on Mining and Learning from Time Series – Deep Forecasting: Models, Interpretability, and Applications, August, 2022.* ACM, Washington, DC, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

Graphs are popular data structures that are effective at compactly representing relationships between entities in structured domains. A rich body of literature have recently explored graph-based representation learning for a range of applications in the context of time-series forecasting — see Feng et al. [2019]; Li et al. [2018]; Mas [2021]; Pareja et al. [2020]; Shumovskaia et al. [2021]; Wu et al. [2019, 2020]; Yu et al. [2018] among others. These works encompass inherently spatial problems such as traffic forecasting [Li et al. 2018; Yu et al. 2018], power networks [Wu et al. 2020], and banking links [Shumovskaia et al. 2021]. However, this same framework also extends to more abstract correlation structures in stock price prediction [Chen et al. 2018], currency exchange rates [Wu et al. 2020], and epidemiology [Mas 2021]. Thus, effective tools in this domain present promising opportunities to advance the state-of-the-art forecasting performance across these and many more applied problems.

These works employ (temporal) graph neural networks (GNNs) that extend convolutional neural networks to irregular graph domains. Most of these works fundamentally rely on a (given) static graph [Li et al. 2018; Wu et al. 2019; Yu et al. 2018]. In many real-world scenarios, the graph structure can evolve over time e.g., social networks [Berger-Wolf and Saia 2006], detecting fraud and crime in financial networks [Pareja et al. 2020], and analyzing contact tracing data [Malik et al. 2021]. It is important to capture such evolutions in graphs into the learnt graph representations for improved generalization. There has been limited work in models that learn from dynamically changing graphs [Malik et al. 2021; Pareja et al. 2020; Piaggesi and Panisson 2022]. These models are either prohibitively slow and/or suffer from poor generalization performance — as our experiments show.

We introduce models that provide superior out-of-sample generalization, while being attractive in terms of computational efficiency. We propose a combination of temporal convolutions on the time-varying adjacency space and Tensor Graph Convolutional Layer (TGCL), which collectively capture dynamic trends in graphs effectively in the time-varying graph representations. The joint module can be combined with existing state-of-the-art (temporal) GNN architectures that have previously relied on only static graphs e.g., Spatio-Temporal Graph Convolutional Networks (STGCN) [Yu et al. 2018], Graph WaveNet (GWN) [Wu et al. 2019], MTGNN [Wu et al. 2020]. We equip our time-varying graph-based models with cheaply computed time-varying partial correlation graphs — when time-varying graphs are not given, but the correlations are naturally time-varying — and demonstrate through our experiments that temporal convolutions on the time-varying adjacency space and TGCL improve the out-of-sample generalization of these models significantly. Interestingly, our TGCL-equipped models are 30% better than prior models that rely on dynamic graphs e.g., EvolveGCN [Pareja et al. 2020] and TM-GCN [Malik et al. 2021]. Additionally, our TGCL-based models are 300× faster to train than the more competitive dynamic graph-based EvolveGCN baseline.

*Contributions.* Our contributions can be summarized as follows: (i) We propose temporal convolutions on the time-varying graph adjacency space, which operates alongside the temporal convolutions on the time-varying feature space. (ii) We propose Tensor Graph Convolutional Layer (TGCL) that generalizes the graph convolutional operator to the tensor space, combining time-varying (learnt) feature representations and time-evolving (learnt) graph representations. (iii) TGCL along with the temporal convolutions on time-varying graphs can be incorporated into many existing temporal GCN architectures e.g., STGCN, GWN. We denote these architectures as Dyn-STGCN, Dyn-GWN and show improved out-of-sample generalization in comparison with their respective (static) counterparts on finance and traffic datasets. (iv) Interestingly, in comparison with state-of-the-art baselines that rely on dynamic graphs e.g., EvolveGCN, TM-GCN, our proposed Dyn-GWN leads to 35% better test performance on traffic forecasting PEMS-BAY dataset. (v) Our Dyn-GWN model is $\sim$ 300× faster than EvolveGCN (more competitive baseline from dynamic graph models).

## 2 RELATED WORK

*Static Spatio-temporal Models.* Recent work has focused on developing models that can leverage a static spatial structure. Yu et al. [2018] approach traffic forecasting from a purely convolutional architecture perspective to propose STGCN, which applies graph convolutions to the graph adjacency and 1D causal convolutions to the temporal axis. Li et al. [2018] leverage the idea of diffusion convolution to capture spatial dependencies in the context of traffic forecasting in their DCRNN architecture. Both these approaches have become foundational in this domain, sparking a series of models, improving upon both spatial and temporal aspects of these models. See Table 1 (and description) in Jiang et al. [2021] for an extensive overview.

Notable among these later models are Graph WaveNet [Wu et al. 2019] and MTGNN [Wu et al. 2020]. Wu et al. [2019] builds on the work of Li et al. [2018] using diffusion convolutions in both a forward and backward direction, as well as learning a "self-adaptive" adjacency matrix to capture hidden dependencies which are not represented in the original adjacency. To capture temporal relationships, they apply 1D causal convolutions because of their ability to capture long-range relationships and scale efficiently. Wu et al. [2020] consider applications where the graph structure is not known a priori. They focus on learning the graph structure from purely time-series data. Their approach uses similar 1D convolutions to the approach by Yu et al. [2018], but also employees MixHop-based [Abu-El-Haija et al. 2019] graph convolution layers which promise greater representation power by learning weights for a mixture of adjacency matrix powers. Both Graph WaveNet and MTGNN models have been independently validated to be the state-of-the-art models in traffic forecasting comparison survey — See Table 5 in Jiang et al. [2021]. None of these models consider temporal convolutions on the dynamic adjacency space (as we propose in this work) along with TGCL-based aggregation scheme. Our proposals can be integrated into these leading models as we demonstrate with two particular models: — STGCN and Graph WaveNet.

**Table 1: Qualitative Comparison of Spatio-temporal Models**

| Model | Graph Type | Training Time | Generalization |
|---|---|---|---|
| DCRNN | Static | Slow | Strong |
| STGCN | Static | Fast | Strong |
| GWN | Static | Fast | Strong |
| EvolveGCN | Dynamic | Slow | Strong |
| TM-GCN | Dynamic | Fast | Poor |
| **Dyn-STGCN**(ours) | Dynamic | Fast | Strong |
| **Dyn-GWN**(ours) | Dynamic | Fast | Strong |

*Dynamic Spatio-temporal Models.* Next, we summarize works that consider dynamic graphs in their modeling approaches. One of the earliest such methods was by Pareja et al. [2020]. With EvolveGCN, they extend the work on graph convolutions done by Kipf and Welling [2016] to dynamic graphs by applying a recurrent model to capture the evolution of the GCN parameters. They model the dynamics of the GCN's weights over time via recurrent architectures, which "evolve" the weights based on the previous weights and/or the current node embeddings. Malik et al. [2021] extend the graph convolution paradigm even further to three-dimensional tensors that explicitly incorporate a time dimension. Their work replicates the functional form of graph convolutions while redefining the convolution function using Tensor-M Products, thus the name, TM-GCN.

*Limitations of existing work.* Although adequate methods exist for spatio-temporal forecasting with static graphs, all existing methods for dynamic graphs are either slow in training and/or inference or provide relatively poor generalization performance. We provide a qualitative summarize in Table 1. Although EvolveGCN [Pareja et al. 2020] shows strong performance on a number of tasks, its recurrent architecture renders this method prohibitively time-consuming to train. As a result, it is challenging to tune to new datasets and tasks. TM-GCN [Malik et al. 2021] is orders of magnitudes more computationally efficient on spatio-temporal datasets. However, it shows poor generalization in our experiments. Thus, we propose Dyn-STGCN and Dyn-GWN as methods, which address both shortcomings in a principled manner, drawing inspiration from TM-GCN [Malik et al. 2021] and STGCN [Yu et al. 2018]/GWN [Wu et al. 2019]. We validate with substantial empirical work relative to a wide range of baselines.

## 3 PROBLEM DEFINITION

We refer to a sequence of graphs: $\mathcal{G}^{(t)} = (\mathcal{V}, \boldsymbol{A}^{(t)}, \boldsymbol{X}^{(t)}),\ t \in \{1, \ldots, T\}$, with a fixed set $\mathcal{V}$ of $N$ nodes, time-varying adjacency matrices $A^{(t)} \in \mathbb{R}^{N \times N}$, and time-varying node feature matrices $X^{(t)} \in \mathbb{R}^{N \times p}$. Note $X^{(t)}_{n,:} \in \mathbb{R}^p$ is the feature vector consisting of $p$ features associated with node $n$ at time $t$. The graphs can be weighted, and directed or undirected. These can be generalized across time. A time-varying graph is represented by a tensor $\mathbf{G} \in \mathbb{R}^{T \times N \times N}$ which concatenates $A^{(t)}$ across all timesteps $t \in [T]$. Similarly, a feature tensor, $\mathbf{X} \in \mathbb{R}^{T \times N \times p}$ concatenates the node features $X^{(t)}$ at each timestep. A static graph setting is represented by $\mathcal{G} = (\mathcal{V}, A, X^{(t)})$, with the same set $\mathcal{V}$ of nodes and fixed adjacency matrix $A \in \mathbb{R}^{N \times N}$ (capturing long-term dependencies) and time-varying node feature matrices $X^{(t)}$.

We consider a dynamic graph representation learning paradigm that aims to learn a function $f(\cdot)$ that maps $T'$ historical graph signals and time-varying graphs to future $h$ graph signals, i.e.,

$$[\mathbf{X}_{(t-T'+1:t)}, \mathbf{G}_{(t-T'+1:t)}, S] \xrightarrow{f(\cdot)} \mathbf{X}_{(t+1:t+h)}, \tag{1}$$

where the dynamic learning paradigm employs additional (given) static graph to capture long-term dependencies. The time-varying graphs are assumed to be apriori known and are passed to the learning problem — this is similar to the learning paradigm studied by Malik et al. [2021]; Pareja et al. [2020].

## 4 PROPOSED ARCHITECTURES: DYN-STGCN AND DYN-GWN

Given the strong generalization performance of prior models with static graphs, STGCN and GWN presents a compelling direction to extend to dynamic graphs. These static models rely on two fundamental blocks: 1D temporal convolutions and graph convolution. The key challenges of catering to dynamic graphs with these models are:

(i) how to extend temporal convolution blocks to tensors, rather than matrices;
(ii) how to aggregate information from learnt dynamic graph embeddings with the learnt feature embeddings.

For the first challenge, we get inspiration from the approach taken for node features, and devise a similar strategy that performs gated temporal convolutions on adjacency tensors. For the second challenge we introduce a Tensor Graph Convolutional Layer (TGCL) . We discuss our proposals in detail in Sections 4.1, and 4.2.

Our proposals are applicable to multiple existing state-of-the-art methods e.g., Spatio-Temporal Graph Convolution Network (STGCN) [Yu et al. 2018], Graph WaveNet (GWN) [Wu et al. 2019], MTGNN [Wu et al. 2020] etc. We demonstrate the applicability of our proposals in the context of STGCN and GWN. We denote our new methods as Dyn-STGCN and Dyn-GWN.

### 4.1 Gated Temporal Convolution Layers with Residual Connections

We first describe the gated temporal convolutional layers with residual connections that operate on the time-varying adjacency graph tensors. These layers consist of three components: dilated causal convolutions [Yu and Koltun 2016], gating [Dauphin et al. 2017], and residual connections [He et al. 2016]. These gated temporal convolution layers capture temporal trends in the dynamic graphs. In prior work, such modules have been used to capture temporal trends in node features only, e.g., in Wu et al. [2019]. However, the temporal trends in dynamic graphs also capture useful information as we show in our work. Next, we describe individual components of the gated temporal convolutional layers.

*Dilated Causal Convolutions.* Dilated causal convolutions have multiple favorable characteristics:

(i) they allow an exponentially large receptive field by increasing the layer depth,
(ii) they can support long-range sequences,
(iii) they allow efficient parallel computation because they do not require recursive computation,
(iv) they allow causal prediction (prediction does not rely on future time steps) by zero padding.

As a special case of standard 1D convolution, the dilated causal convolution operation slides over inputs by skipping values with a certain step. Mathematically, given a 1D sequence input $x \in \mathbb{R}^{T'}$ and a filter $h_\theta \in \mathbb{R}^K$, the dilated causal convolution operation of $x$ with $h_\theta$ at step $t$ is represented as

$$\boldsymbol{x} \star \boldsymbol{h}_\theta(t) = \sum_{s=0}^{K-1} \boldsymbol{h}_\theta(s)\boldsymbol{x}(t - d \times s), \tag{2}$$

where $d$ is the dilation factor which controls the skipping distance.

*Gated Temporal Convolution Network.* Gating mechanisms have shown promising results in both recurrent and temporal architectures [Dauphin et al. 2017]. A simple Gated Temporal Convolution Network (Gated TCN) takes the form:

$$h = g(\boldsymbol{x} \star h_{\theta_1}) \odot \sigma(\boldsymbol{x} \star h_{\theta_2}), \tag{3}$$

where $x$ is the input, $\theta_1$, $\theta_2$ are learnable parameters, $\odot$ is the element-wise Hadamard product, $g(\cdot)$ is an activation function, and $\sigma(\cdot)$ is the sigmoid function which determines the ratio of information passed to the next layer. Note there are additional bias terms, which we omit here for convenience. We adopt Gated TCN in our model to learn complex temporal dependencies. Empirically, we used tangent hyperbolic function ($tanh(\cdot)$) as the activation function $g(\cdot)$. In tensor notation, we denote this operation as $\Gamma \star_{\mathbf{X}} \Theta$.

*Residual Connections.* Given the success of residual connections in various neural network architectures, e.g. ResNets [He et al. 2016], graph convolutional networks [Chen et al. 2020] etc., we include additional residual connection in the gated TCN as follows:

$$h = \left(g\left(\boldsymbol{x} \star h_{\theta_1}\right) + \boldsymbol{x}^{res}\right) \odot \sigma(\boldsymbol{x} \star h_{\theta_2}), \tag{4}$$

where $x^{res}$ denotes the residual connection from the prior TCN layer. In tensor notation, we denote this operation as $\Gamma_{res} \star_{\mathbf{X}} \Theta$.

*4.1.1 Application of Gated TCN with residual connections to dynamic graphs.* We propose to apply the operation in (4) on the dynamic graphs. This function operates in parallel to the (existing) temporal convolutional layers (operating on signals/features) in STGCN and GWN architectures. Stacking of gated TCN layers allows learning of longer term dependencies in both the dynamic graphs and the dynamic node signals — the existing STGCN and GWN only learn longer term dependencies in dynamic node signals.

Application of the operation in (4) requires some implementation considerations. In order to make use of existing efficient

GPU-compatible temporal convolutional implementations in popular deep learning API, we pass the reshaped version of the dynamic graphs to the modules. **G** is processed (at the input) into a shape: $(b, T', N^2, 1)$, where $b$ represent the batch-size, $T'$ denotes the historical lag, $N^2$ denotes all the pair-wise entries in the graphs. The successive Gated TCN modules generate $\mathbf{G}^l \in \mathbb{R}^{b,T'-d_{[l]}*(K-1),N^2,c^l_{out}}$, where $c^l_{out}$ denotes the number of latent dynamic graphs and $d_{[l]}$ denotes dilation factors up to layer $l$. Note $c^{l-1}_{out} = c^l_{in}$ for $l \in [L]$, $c^1_{in} = 1$, and $L$ denotes the number of successive gated TCN modules in the architecture.

*Adjacency Normalization Layer.* After empirical investigations, we found that properly normalizing adjacency tensors is essential to strong performance with our model. In particular, we transform each latent (or input) adjacency slice into $[0,1]^{N\times N}$. This is only necessary before passing the adjacency tensor to TGCL, which is described in Section 4.2. Thus, we construct the Adjacency Normalization Layer, denoted by the function, $\bar{\mathbf{G}}^l = n(\mathbf{G}^l)$. We take the output of the Gated TCN with residual connections on the adjacency space from layer $l$: $\mathbf{G}^l \in \mathbb{R}^{b,T'-d_{[l]}*(K-1),N^2,c^l_{out}}$, as an input and transform it into $\bar{\mathbf{G}}^l$ as follows:

$$\bar{\mathbf{G}}^l = n(\mathbf{G}^l) = \text{softmax}(\mathbf{G}^l_r) \tag{5}$$

where $\mathbf{G}^l_r \in \mathbb{R}^{b,T'-d_{[l]}*(K-1),N,N,c^l_{out}}$ is a reshaped version of $\mathbf{G}^l$, and softmax$(\cdot)$ is applied on the second to last dimension of $\mathbf{G}^l_r$. Softmax normalization has been used in prior work in static graph learning literature — see Wu et al. [2019] among others. The normalization ensures that the latent graphs have non-negative weights when they are used in the graph convolution/aggregation operation. As pointed out by Derr et al. [2018], the aggregation methods in graph convolution operation in GCNs is traditionally designed for unsigned graphs because they assumes social theory homophily. Signed graphs require more complex aggregation methods (see Derr et al. [2018] for an example) — we do not pursue signed graphs in this work.

*Relation of Temporal Convolution on adjacency space to the approach in TM-GCN [Malik et al. 2021].* Conceptually, our proposal to apply temporal convolutions on the dynamic graphs is motivated by the approach taken by Malik et al. [2021]. They apply the tensor M-product framework [Kernfeld et al. 2015; Kilmer et al. 2013] to compute an average on the adjacency tensor over the temporal dimension. By applying gated temporal convolutions, we achieve a similar weighted average in the non-linear space.

## 4.2 Tensor Graph Convolutional Layer

Graph convolution is an essential operation to extract a node's features given its structural information. The two popular approaches in existing literature are summarized below. The first approach smooths a node's signal by aggregating and transforming its neighborhood information through first approximations of Chebyshev filters [Defferrard et al. 2016; Kipf and Welling 2016]. This has been used in STGCN [Yu et al. 2018] among others. The second approach models the diffusion process of graph signals with $V$ steps. This approach was proposed by the authors of DCRNN [Li et al. 2018] and also later used by authors of Graph WaveNet [Wu et al. 2019] in the context of learning from a static graph. We use the second approach in the context of learning from dynamic graphs.

In the context of dynamic graphs, applying graph convolution operation requires a careful consideration of how to combine the latent temporal features with the latent temporal graphs. This is different from the static setting as the same graphs is combined across all channels of the latent temporal features. A naive parameterization would consider an outer product of the latent temporal features with the latent temporal graphs. However, this results in substantial overparameterization. To prevent this overparameterization, our parallel architecture with temporal convolutional layers on time-varying graphs (described in Section 4.1) matches the number of channels per layer with that of the temporal convolutional layers (operating on the time-varying features). This makes for a compact paramaterization. We combine this parameterization with the diffusion process approach for graph convolution and write the tensor graph convolution operation:

$$\left(\mathbf{H}^l \star_{\mathbf{G}^l} W^l\right)_{bijm} = \sum_{v=1}^{V}\left(\sum_{n=1}^{N}\left(\left(\bar{\mathbf{G}}^l\right)^v\right)_{bijno}\mathbf{H}^l_{bino}\right)\cdot \boldsymbol{W}^{l,v}_{om} \tag{6}$$

where $\left(\bar{\mathbf{G}}^l\right)^v$ denotes the power series of the transition matrix $\bar{\mathbf{G}}^l$, $\mathbf{H}^l$ denotes the latent temporal features at layer $l$, $\boldsymbol{W}^{l,v} \in \mathbb{R}^{c^l_{in},c^l_{out}}$ denotes the learnable matrix for power $v$ for layer $l$. In the case of a directed graph, the diffusion process have both forward and backward directions, which our implementation supports. The output from the TGCL, in our Dyn-STGCN and Dyn-GWN architectures, is aggregated with the outputs of the original graph convolutional layers (operating on static graphs in the original STGCN and GWN architectures).

## 4.3 Complete architectures: Dyn-STGCN and Dyn-GWN

Next, we summarize the two proposed Dyn-STGCN and Dyn-GWN architectures that combine the temporal convolutions on dynamic graphs and TGCL. We visualize the architecture for Dyn-GWN in Fig. 1.

*Dyn-STGCN.* We begin with a normalized (and processed) dynamic graph tensor, **G**, with shape, $(b, T', N^2, 1)$, where $b$ is the batch size, $T'$ is the historical lag, and $N$ is the number of nodes. Now, we can apply the temporal convolution layer directly in adjacency space. Thus, we can represent the full Spatio-temporal Convolution Block in Dyn-STGCN as:

$$\mathbf{H}^l_f = \Gamma \star_{\mathbf{H}^l} \Theta^l_1 \qquad \text{(as in STGCN)} \tag{7}$$

$$\mathbf{G}^l_a = \Gamma_{res} \star_{\mathbf{G}^l} \Theta^l_2 \tag{8}$$

$$\bar{\mathbf{G}}^l = n(\mathbf{G}^l_a) \tag{9}$$

$$\mathbf{Z}^l_{Static} = \hat{A}\mathbf{H}^l_f \boldsymbol{W}^l_{Static} \qquad \text{(as in STGCN)} \tag{10}$$

$$\mathbf{Z}^l_{Dyn} = \mathbf{H}^l_f \star_{\bar{\mathbf{G}}^l} \boldsymbol{W}^l_{Dyn} \tag{11}$$

$$\mathbf{Z}^l = \text{ReLU}\left(\mathbf{Z}^l_{Static} + \mathbf{Z}^l_{Dyn} + \mathbf{H}^{l,res}\right) \tag{12}$$

$$\mathbf{H}^{l+1} = \Gamma \star_{\mathbf{Z}^l} \Theta^l_3 \qquad \text{(as in STGCN)} \tag{13}$$

$$\mathbf{G}^{l+1} = \Gamma^l_{res} \star_{\mathbf{G}^l_a} \Theta^l_4 \tag{14}$$

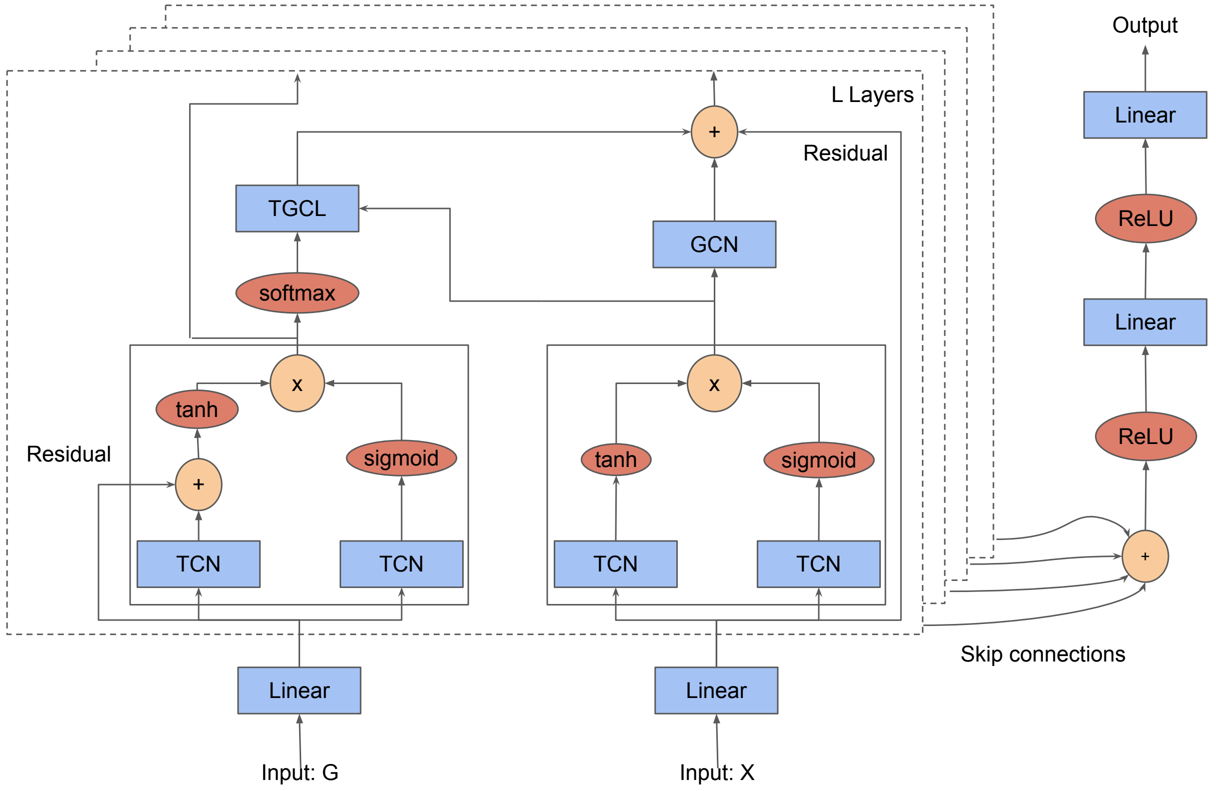

**Figure 1: Illustration of Dynamic Graph WaveNet (Dyn-GWN).**

where $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, $\tilde{A} = A + I_N$, $I_N \in \mathbb{R}^{N,N}$ is an identity matrix, and $\tilde{D}$ is a diagonal matrix with diagonal entries $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. $\mathbf{G}^l \in \mathbb{R}^{b,T'-2l(K-1),N,N,c^l_{in}}$ and $\mathbf{H}^l \in \mathbb{R}^{b,T'-2l(K-1),N,c^l_{in}}$ are layer $l$ inputs, producing outputs $\mathbf{G}^{l+1} \in \mathbb{R}^{b,T'-2(l+1)(K-1),N^2,c^l_{out}}$ and $\mathbf{H}^{l+1} \in \mathbb{R}^{b,T'-2(l+1)(K-1),N,c^l_{out}}$. Note that the dilation rate is 1 in all temporal convolutional layers in Dyn-STGCN (as in STGCN). $\mathbf{H}^{l+1}$ is also passed through batch normalization and dropout layers in order to stabilize learning before being passed to the next block.

*Dyn-GWN.* We begin with a normalized (and processed) dynamic graph tensor, $\mathbf{G}$, with shape, $(b, T', N^2, 1)$, where $b$ is the batch size, $T'$ is the historical lag, and $N$ is the number of nodes. Now, we can apply the temporal convolution layer directly in adjacency space. Thus, we can represent each module in Dyn-GWN as:

$$\mathbf{H}^l_f = \Gamma \star_{\mathbf{H}^l} \Theta^l_1 \quad \text{(as in GWN)} \tag{15}$$

$$\mathbf{G}^l_a = \Gamma_{res} \star_{\mathbf{G}^l} \Theta^l_2 \tag{16}$$

$$\bar{\mathbf{G}}^l = n(\mathbf{G}^l_a) \tag{17}$$

$$\mathbf{S}^l_{Static} = \sum_{v=0}^{V} \overline{A}^v \mathbf{H}^l_f W_{v,1} + \left(\overline{A^T}\right)^v \mathbf{H}^l_f W_{v,2} \quad \text{(as in GWN)} \tag{18}$$

$$\mathbf{S}^l_{adp} = \sum_{v=0}^{V} A^v_{adp} \mathbf{H}^l_f W_{v,3} \quad \text{(as in GWN)} \tag{19}$$

$$\mathbf{S}^l_{Dyn} = \mathbf{H}^l_f \star_{\bar{\mathbf{G}}^l} W^l_{Dyn} \tag{20}$$

$$\mathbf{H}^{l+1} = \mathbf{S}^l_{Static} + \mathbf{S}^l_{adp} + \mathbf{S}^l_{Dyn} + \mathbf{H}^{l,res} \tag{21}$$

$$\mathbf{G}^{l+1} = \mathbf{G}^l_a \tag{22}$$

where $\overline{A}$ is a row-sum normalized version of $A$ and $\overline{A^T}$ is a row-sum normalized version of $A^T$. $\mathbf{G}^l \in \mathbb{R}^{b,T'-d_{[l]}(K-1),N,N,c^l_{in}}$ and $\mathbf{H}^l \in \mathbb{R}^{b,T'-d_{[l]}(K-1),N,c^l_{in}}$ are layer $l$ inputs, producing outputs $\mathbf{G}^{l+1} \in \mathbb{R}^{b,T'-d_{[l+1]}(K-1),N^2,c^l_{out}}$ and $\mathbf{H}^{l+1} \in \mathbb{R}^{b,T'-d_{[l+1]}(K-1),N,c^l_{out}}$. $\mathbf{H}^{l+1}$ is also passed through dropout layers before being passed to the next block in order to reduce overfitting. Note that the dilation rate $d_{[l]}$ denotes the dilation rate up to layer $l$, which controls the reduction in temporal dimension as successive temporal convolutions are applied. The module in Dyn-GWN is shown in Fig. 1.

## 5 DATASETS

### 5.1 Financial Market Datasets: Stock Volatilities

We evaluate the time-series forecasting with dynamic graphs in the context of stock volatilities. Volatilities prediction has been studied by multiple works [Barigozzi and Brownlees 2019a; Diebold and Yilmaz 2014, 2015; Engle et al. 2012; Ibrahim et al. 2021]. We consider 80 companies from S&P500 financial market and define the daily volatility, as given in Barigozzi and Brownlees [2019a]; Diebold and Yilmaz [2015]; Ibrahim et al. [2021]; Parkinson [1980], using the daily high and low stock prices:

$$\tilde{\sigma}^2_{it} = 0.361 \left(\log p^{\text{high}}_{it} - \log p^{\text{low}}_{it}\right)^2 \tag{23}$$

where $p_{it}^{\text{high}}$ and $p_{it}^{\text{low}}$ denote the maximum and minimum price of stock $i$ on day $t$.

*5.1.1 Choice of Dynamic Graphs.* We consider two different dynamic graphs for this dataset: (i) EDGAR Cosearch Graphs (ii) Partial Correlation Graphs. While the former are recovered (and processed) from an alternate data source, the latter are cheaply computed from the same time-series data to cater to settings when dynamic graphs are not easily available. We summarize both these approaches below.

*Dynamic Graphs: EDGAR Cosearch Graphs.* Based on a paper by Lee et al. [2015], the EDGAR cosearch dataset builds a graph from stocks searched in SEC filings. Each company represents a node, while edge weights are computed by renormalizing the number of times that company A is searched for immediately before company B. This weight corresponds to a directed edge from A to B. We leverage this weighted connectivity matrix to predict time-series targets such as returns, volatilities, and volumes that are derived from Yahoo Finance during the timeframe of the EDGAR dataset. The experiments here focus on on the task of predicting daily volatilities from the EDGAR cosearch graph with lagged daily volatilities as the node features. The node features for each are smoothed by averaging over the preceding 48 steps using a sliding window approach. We are using graphs from 2005 to 2015 in the training set, 2015 to 2016 in the validation set, and 2016 to 2017 in the testing set. For methods that only accommodate a static graph, we compute an average adjacency over the entire time period.

*Dynamic Graphs: Partial Correlations.* To further explore the time-series from the Yahoo Finance data, we construct new graphs to directly capture the time-series correlation. Beginning with the raw pairwise time-series, we compute the Ledoit-Wolf covariance matrix [Ledoit and Wolf 2004] between all companies in the subset of interest using a rolling window of fixed size. We empirically find that a window size of 48 is optimal with the SP500 data. To construct the adjacency, we use matrix inversion to compute the corresponding precision matrix. With this in hand, we compute partial correlation from the previous matrix as described by Barigozzi and Brownlees [2019b] and take the absolute value. This provides a non-negative correlation structure between companies at each timestep. For the experiments leveraging these graphs, we use identical data to the original SP500 and EDGAR dataset above and simply replace the EDGAR graphs with those constructed using partial correlation.

## 5.2 Traffic Datasets: METR-LA & PEMS-BAY

As traffic forecasting is a common application of spatio-temporal methods, we also explore the performance of our model in this context. We conduct experiments on two datasets used by Li et al. [2018]; Wu et al. [2019]. These are METR-LA and PEMS-BAY. METR-LA contains traffic data from loop detectors in the highway of Los Angeles County (Jagadish et al., 2014). We select 80 sensors and sample an approximately week long chunk of data for our experiments. We apply the same methodology to the PEMS-BAY data. PEMS-BAY dataset was produced by California Transportation Agencies (CalTrans) Performance Measurement System (PeMS). For both datasets, we used the validation and test splits used by Wu et al. [2019]. We used the last 20,000 training samples for METR-LA and 5000 training samples for PEMS-BAY from the original training sets.

*Dynamic Graphs: Partial Correlations.* To further validate the robustness of our model, we apply the same partial correlation-based method to these two datasets. For static graph methods, we use the original spatial adjacency provided by Li et al. [2018].

## 6 EXPERIMENTS

This section demonstrates validity of temporal convolutions on the dynamic adjacency space and the TGCL aggregation scheme on time-series problems with extensive sets of experiments. For experiments, we employ state-of-the-art models such as STGCN [Yu et al. 2018] and GWN [Wu et al. 2019] and optimize the models with proposed modifications. Note that we use a notation of Dyn-model to indicate models optimized with our proposals.

We study the performance of our models in various settings and compare against the relevant state-of-the-art baselines for each setting. The different settings can be summarized as follows:

(i) Comparison of Dyn-STGCN against (static) STGCN and classical high-dimensional statistical approaches for time-series forecasting e.g., vector autoregression (VAR) [Sims 1980] and multioutput Random Forests (RF) [Breiman 2001]. We consider mean absolute errors (MAE) and mean squared errors (MSE) as evaluation metrics. We consider two types of external dynamic graphs: EDGAR and Partial Correlations.

(ii) Comparison of Dyn-GWN against (static) GWN on METR-LA for multi-step horizon forecasting (up to 12). We consider mean absolute errors (MAE), root mean squared errors (RMSE), and mean absolute percentage errors (MAPE). Following prior work, different horizon steps $\{3, 6, 9\}$ were considered. We also include average performance across 12 horizons.

(iii) Comparison of Dyn-GWN against dynamic spatio-temporal models: EvolveGCN and TM-GCN on PEMS-BAY for multi-step horizon forecasting. We again consider MAE, RMSE and MAPE for different horizons. In addition, we also compare training and inference times of Dyn-GWN against those of EvolveGCN (more promising than TM-GCN in our experiments).

*Models Implementation.* We implemented our models in Tensorflow and PyTorch. We implemented Dyn-STGCN model in Tensorflow and Dyn-GWN in PyTorch.

*Computing Setup.* All experiments are conducted on a Linux cluster (CPU: IBM POWER9 @ 2.90 GHz, GPU: Nvidia Tesla V100).

## 6.1 Case Study: Dyn-STGCN on Stock Volatility

We consider daily stock volatility dataset described in Section 5.1 with dynamic EDGAR graphs and dynamic partial correlation graphs. We study the performance of Dyn-STGCN in this setting with the two dynamic graphs.

*Competing Methods.* We compare Dyn-STGCN against (static) STGCN and classical high-dimensional statistical approaches for time-series forecasting e.g., vector autoregression (VAR) [Sims 1980] and multioutput Random Forests (RF) [Breiman 2001]. We used

**Table 2: Test Set Performance of Dyn-STGCN on stock volatility dataset with two dynamic graphs (EDGAR and Partial Correlations). Hyphen (-) indicates the method does not use static and/or dynamic graph.**

| Model | Static Graph | Dynamic Graph | MSE | MAE |
|---|---|---|---|---|
| RF | - | - | 0.1313 ± 0.0001 | 0.2908 ± 0.0001 |
| VAR | - | - | 0.1168 ± 0.0000 | 0.2719 ± 0.0000 |
| STGCN | EDGAR | - | 0.1066 ± 0.0026 | 0.2558 ± 0.0030 |
| Dyn-STGCN | EDGAR | EDGAR | **0.1038** ± 0.0005 | **0.2522** ± 0.0007 |
| Dyn-STGCN | EDGAR | Partial Correlation | **0.0716** ± 0.0012 | **0.2096** ± 0.0018 |

Ridge regressor in scikit-learn [Buitinck et al. 2013] for setting up multivariate VAR with ridge penalty. We used multioutput Random Forests from scikit-learn [Buitinck et al. 2013].

*Tuning.* We perform 1000 trials of hyperparameter tuning with a random search (via hyperopt [Bergstra et al. 2013]). All models are tuned with respect to historical lag in the range $[5, 100]$. For VAR, $\alpha$ was sampled uniformly on the log scale in the range $[10, 10^6]$. For RF, we tuned over number of trees in the range $[1, 100]$, depths over $[2 - 20]$, minimum samples for split in the set $\{1, 2, 4, 6, 8\}$ and minimum samples per leaf in the range $[1, 20]$. For STGCN and Dyn-STGCN, we used a single STGCN block with fixed channels: $[8, 32]$. We tuned over learning rates in the range $[10^{-3}, 10^{-1}]$, batch sizes over the set $\{16, 32\}$, dropout in the range $[0.03, 0.1]$. Both these models were run for 500 epochs with early stopping (patience=10) based on validation set.

*Evaluation Metrics.* We train the models with mean squared error objective. To evaluate the performance of the models studied, we use Mean Squared Error (MSE) and Mean Absolute Error (MAE). After tuning, we train each model for 50 repetitions (using random initialization) and report the averaged results along with their standard errors. We report results for both metrics for all models across 50 trials.

*Results.* Table 2 shows the results of Dyn-STGCN and baselines for stock volatility prediction. Clearly, there are benefits from incorporating dynamic graph structures into these prediction tasks. Our proposed model outperforms Dyn-STGCN all baselines with both types of dynamic graphs in both evaluation metrics. Notably, the cheaply computed dynamic partial correlation graphs outperform the externally generated dynamic EDGAR graphs.

## 6.2 Case Study: Dyn-GWN on METR-LA

We compare performance of Dyn-GWN against (static) GWN on real-world METR-LA traffic dataset — See Section 5.2 for more details. We use dynamic partial correlation graphs for Dyn-GWN. We trained both models with (masked) mean absolute error objective.

*Evaluation Metrics.* These methods are evaluated based on three commonly used metrics in traffic forecasting, including (i) Mean Absolute Error (MAE), (ii) Root Mean Squared Error (RMSE), and (iii) Mean Absolute Percentage Error (MAPE). Missing values are excluded in calculating these metrics as done by earlier works [Li et al. 2018; Wu et al. 2019].

*Tuning.* The architectures were fixed to 4 blocks for both models. We used a fixed batch size of 64 and ran the models for 200 epochs with early stopping (patience=25) based on a validation set. We tuned over learning rates in the range $[0.0001, 0.01]$. We report the performance on held-out test set.

*Results.* Table 3 shows the comparison of different approaches for 15 minutes (H3), 30 minutes (H6), 45 minutes (H9) and average across 5 minute to 1 hour (H1-12) horizon forecasting. We can observe a clear advantage of using dynamic graphs. Dyn-GWN outperforms GWN across all metrics and all horizons, which highlights the importance of capturing the evolution in the graph adjacency space.

**Table 3: Comparison of Dyn-GWN with (static) GWN on METR-LA dataset for multi-step horizon forecasting**

| | | Multi-step horizon forecast | | | |
|---|---|---|---|---|---|
| Metric | Model | H3 | H6 | H9 | H1-12 (average) |
| RMSE | GWN | 5.3407 | 6.3202 | 6.8835 | 6.1490 |
| | Dyn-GWN | **5.3069** | **6.2715** | **6.7967** | **6.0943** |
| MAE | GWN | 2.8212 | 3.1962 | 3.4264 | 3.1420 |
| | Dyn-GWN | **2.8139** | **3.1863** | **3.4053** | **3.1342** |
| MAPE | GWN | 7.3400 | 8.6800 | 9.3900 | 8.4300 |
| | Dyn-GWN | **7.3200** | **8.5800** | **9.2800** | **8.3600** |

## 6.3 Case Study: Dyn-GWN vs EvolveGCN vs TM-GCN on PEMS-BAY

We compare performance of Dyn-GWN against dynamic graph representation learning methods on real-world PEMS-BAY traffic dataset — See Section 5.2 for more details. We use dynamic partial correlation graphs for all methods. We trained all models with (masked) mean absolute error objective. These methods are again evaluated based on (masked) MAE, RMSE, and MAPE.

*Competing Methods.* We consider EvolveGCN [Pareja et al. 2020] and TM-GCN [Malik et al. 2021] models. The original implementations provided by the authors were updated to support training with (masked) mean absolute error loss.

*Tuning.* The architectures were fixed for EvolveGCN and TM-GCN to default settings. For Dyn-GWN, we used 2 blocks. For a fair comparison against EvolveGCN, which is prohibitively slow, we performed only 20 tuning trials for all three models (Dyn-GWN, EvolveGCN, TM-GCN) that tuned over learning rates in the range

$[0.0001, 0.01]$. We capped the optimization time per hyperparameter trial for EvolveGCN to 5 days (108 hours). In comparison, the average convergence time per-trial for Dyn-GWN is 2.2 hours.

*Results.* Table 4 shows the comparison of different approaches for 15 minutes (H3), 30 minutes (H6), 45 minutes (H9) and average across 5 minute to 1 hour (H1-12) horizon forecasting. We can observe a clear advantage of Dyn-GWN over the competing methods. Dyn-GWN outperforms EvolveGCN and TM-GCN across all metrics and all horizons with a large margin.

**Table 4: Comparison of Dyn-GWN with dynamic graph methods on PEMS-BAY dataset for multi-step horizon forecasting**

| | | Multi-step horizon forecast | | | |
|---|---|---|---|---|---|
| Metric | Model | H3 | H6 | H9 | H1-12 (average) |
| RMSE | TM-GCN | 9.7217 | 10.2726 | 10.8323 | 10.3423 |
| | EvolveGCN | 5.4819 | 6.1412 | 6.6995 | 6.1737 |
| | Dyn-GWN | **3.0463** | **4.2872** | **4.8884** | **4.0297** |
| MAE | TM-GCN | 4.4414 | 4.6637 | 4.8809 | 4.6906 |
| | EvolveGCN | 4.3084 | 4.6456 | 4.9615 | 4.6800 |
| | Dyn-GWN | **1.4302** | **1.8667** | **2.0991** | **1.7866** |
| MAPE | TM-GCN | 13.120 | 13.720 | 14.310 | 13.800 |
| | EvolveGCN | 8.768 | 9.701 | 10.592 | 9.815 |
| | Dyn-GWN | **3.220** | **4.620** | **5.430** | **4.420** |

*Timing comparison with EvolveGCN.* We compare the time of our Dyn-GWN model with time-varying graph-based methods. In particular, we compare against EvolveGCN [Pareja et al. 2020], which has more competitive performance (than TM-GCN [Malik et al. 2021]). We consider both training and inference times in our comparison. We train both Dyn-GWN model for 1 epoch on a Tesla V100 GPU. The number of time samples for training and inference are taken to be 5000. Both models are run with batch-size equal to 64. We show the timing in seconds in Table 5. We can observe that Dyn-GWN is 300× faster in terms of training time and 2000× faster in terms of inference.

**Table 5: Timing comparison of Dyn-GWNet against EvolveGCN for 1 epoch with 64 batch size, $T_{train} \approx 5000$, $T_{valid} \approx 5000$, and 80 nodes on Tesla V100 GPU. Our Dyn-GWN is 300× faster than EvolveGCN.**

| Model | Training Time | Inference Time |
|---|---|---|
| EvolveGCN | 16400s | 15400s |
| Dyn-GWN | 52s | 8s |

## 7 CONCLUSION

To summarize, we propose a new framework for spatio-temporal learning from dynamic graphs. The two main components of our models are: (i) temporal convolution on the time-varying adjacency space, (ii) Tensor Graph Convolutional Layer (`TGCL`) which aggregates latent temporal representations of time-varying node features and time-varying graphs. We generalize previous models to leverage both dynamic and static graphs while maintaining a computationally efficient convolution-based architecture. We demonstrate our proposals in with two new time-varying graph-based methods Dyn-STGCN and Dyn-GWN for time-series forecasting. Experiments demonstrate the efficacy of these model across a range of datasets and tasks. Interestingly, our Dyn-STGCN and Dyn-GCN models are superior at handling dynamic graphs than existing state-of-the-art time-varying graph-based methods e.g., EvolveGCN and TM-GCN in terms of strong generalization while providing both efficient training and inference. In future, we plan to apply this methodology to new forecasting tasks e.g., link prediction.

## ACKNOWLEDGMENTS

This work is supported by the MIT-IBM Watson AI Lab. The views and conclusions are those of the authors and should not be interpreted as representing the official policies of the funding agencies. The authors acknowledge Yada Zhu (IBM Research) and Wenyu Chen (MIT) for discussions. The authors acknowledge the use of MIT Satori Cluster and MIT SuperCloud resources that have contributed to the research results reported within this paper.

## REFERENCES

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 21–29. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/abu-el-haija19a.html.

M. Barigozzi and C. Brownlees. Nets: Network estimation for time series. *Journal of Applied Econometrics*, 34(3):347–364, 2019a.

Matteo Barigozzi and Christian Brownlees. Nets: Network estimation for time series. *Journal of Applied Econometrics*, 34(3):347–364, 2019b. doi: https://doi.org/10.1002/jae.2676. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jae.2676.

Tanya Y. Berger-Wolf and Jared Saia. A framework for analysis of dynamic social networks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 523–528, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402.1150462. URL https://doi.org/10.1145/1150402.1150462.

James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/bergstra13.html.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, et al. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1725–1735. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/chen20v.html.

Yingmei Chen, Zhongyu Wei, and Xuanjing Huang. Incorporating corporation relationship via graph convolutional neural networks for stock price prediction. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 1655–1658, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360142. doi: 10.1145/3269206.3269269. URL https://doi.org/10.1145/3269206.3269269.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 933–941. JMLR.org, 2017.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th*

*International Conference on Neural Information Processing Systems*, NIPS'16, page 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Tyler Derr, Yao Ma, and Jiliang Tang. Signed graph convolutional networks. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 929–934, 2018.

F. X. Diebold and K. Yilmaz. On the network topology of variance decompositions: Measuring the connectedness of financial firms. *Journal of Econometrics*, 182(1): 119–134, September 2014.

Francis X. Diebold and Kamil Yilmaz. *Financial and Macroeconomic Connectedness*. Oxford University Press, March 2015. doi: 10.1093/acprof:oso/9780199338290.001.0001. URL https://doi.org/10.1093/acprof:oso/9780199338290.001.0001.

R. Engle, G. Gallo, and M. Velucchi. Volatility spillovers in east asian financial markets: A mem-based approach. *Review of Economics and Statistics - REV ECON STATIST*, 0: 222–223, 02 2012.

F. Feng, X. He, X. Wang, C. Luo, Y. Liu, and T. Chua. Temporal relational ranking for stock predictions. *ACM Transactions on Information Systems (TOIS)*, 37(2), 2019.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Shibal Ibrahim, Wenyu Chen, Yada Zhu, et al. Knowledge graph guided simultaneous forecasting and network learning for multivariate financial time series. In *Workshop on Machine Learning in Finance*, 2021.

Renhe Jiang, Du Yin, Zhaonan Wang, Yizhuo Wang, Jiewen Deng, Hangchen Liu, Zekun Cai, Jinliang Deng, Xuan Song, and Ryosuke Shibasaki. Dl-traff: Survey and benchmark of deep learning models for urban traffic prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4515–4525, 2021.

Eric Kernfeld, Misha Kilmer, and Shuchin Aeron. Tensor–tensor products with invertible linear transforms. *Linear Algebra and its Applications*, 485:545–570, 2015. ISSN 0024-3795. doi: https://doi.org/10.1016/j.laa.2015.07.021. URL https://www.sciencedirect.com/science/article/pii/S0024379515004358.

Misha E. Kilmer, Karen Braman, Ning Hao, and Randy C. Hoover. Third-order tensors as operators on matrices: A theoretical and computational framework with applications in imaging. *SIAM Journal on Matrix Analysis and Applications*, 34(1):148–172, 2013. doi: 10.1137/110837711. URL https://doi.org/10.1137/110837711.

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

Olivier Ledoit and Michael Wolf. A well-conditioned estimator for large-dimensional covariance matrices. *Journal of Multivariate Analysis*, 88(2):365–411, 2004. ISSN 0047-259X. doi: https://doi.org/10.1016/S0047-259X(03)00096-4. URL https://www.sciencedirect.com/science/article/pii/S0047259X03000964.

Charles M.C. Lee, Paul Ma, and Charles C.Y. Wang. Search-based peer firms: Aggregating investor perceptions through internet co-searches. *Journal of Financial Economics*, 116(2):410–431, 2015. ISSN 0304-405X. doi: https://doi.org/10.1016/j.jfineco.2015.02.003. URL https://www.sciencedirect.com/science/article/pii/S0304405X15000197.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SJiHXGWAZ.

Osman Asif Malik, Shashanka Ubaru, Lior Horesh, Misha E. Kilmer, and Haim Avron. Dynamic graph convolutional networks using the tensor m-product. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 729–737. Society for Industrial and Applied Mathematics, January 2021. doi: 10.1137/1.9781611976700.82. URL https://doi.org/10.1137/1.9781611976700.82.

Jean-François Mas. Spatio-temporal dataset of covid-19 outbreak in mexico. *Data in brief*, 35:106843–106843, 04 2021. doi: 10.1016/j.dib.2021.106843. URL https://pubmed.ncbi.nlm.nih.gov/33589875.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

M. Parkinson. The extreme value method for estimating the variance of the rate of return. *The Journal of Business*, 53(1):61–65, 1980. ISSN 00219398, 15375374.

Simone Piaggesi and André Panisson. Time-varying graph representation learning via higher-order skip-gram with negative sampling. *EPJ Data Science*, 11(1):33, 2022.

Valentina Shumovskaia, Kirill Fedyanin, Ivan Sukharev, Dmitry Berestnev, and Maxim Panov. Linking bank clients using graph neural networks powered by rich transactional data. *International Journal of Data Science and Analytics*, 12(2):135–145, 2021. doi: 10.1007/s41060-021-00247-3. URL https://doi.org/10.1007/s41060-021-00247-3.

Christopher A. Sims. Macroeconomics and reality. *Econometrica*, 48(1):1–48, 1980. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1912017.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. *CoRR*, abs/1906.00121, 2019. URL http://arxiv.org/abs/1906.00121.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks, 2020.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.07122.