Graph Macro Dynamics with Self-Attention for Event Detection

Mert Kosan[1], **Arlei Silva**[2], **Sourav Medya**[3], **Brian Uzzi**[4], **Ambuj Singh**[1]

[1]University of California, Santa Barbara
[2]Rice University
[3]University of Illinois Chicago
[4]Northwestern University
{mertkosan, ambuj}@ucsb.edu, arlei@rice.edu, medya@uic.edu, uzzi@kellogg.northwestern.edu

Abstract

Event detection is a critical task for timely decision-making in graph analytics applications. Despite the recent progress towards deep learning on graphs, event detection on dynamic graphs presents particular challenges to existing architectures. Real-life events are often associated with sudden deviations of the normal behavior of the graph. However, existing approaches for dynamic node embedding are unable to capture the graph-level dynamics related to events. In this paper, we propose DyGED, a simple yet novel deep learning model for event detection on dynamic graphs. DyGED learns correlations between the graph macro dynamics—i.e. a sequence of graph-level representations—and labeled events. Moreover, our approach combines structural and temporal self-attention mechanisms to account for application-specific node and time importances effectively. Our experimental evaluation, using representative datasets, demonstrates that DyGED outperforms competing solutions in terms of accuracy by up to 8.5% while being more scalable than the top alternatives.

Introduction

Event detection on dynamic graphs is a relevant task for effective decision-making in many organizations (Li et al. 2017). In graphs, entities and their interactions are represented as nodes and edges, respectively. The graph dynamics, which changes the interactions and attributes over time, can be represented as a sequence of snapshots. Events, identified as snapshot labels, are associated with a short-lived deviation from normal behavior in the graph.

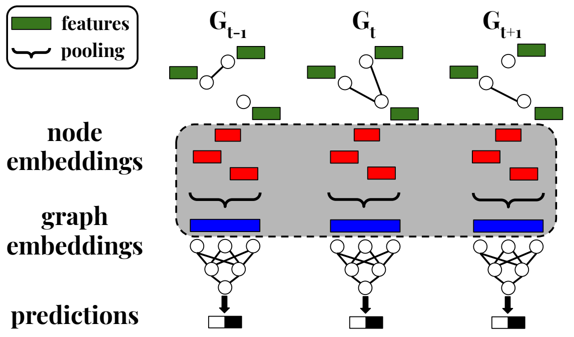
As an example, consider the communication inside an organization, such as instant messages and phone calls (Romero, Uzzi, and Kleinberg 2016). Can the evolution of communication patterns reveal the rise of important events—e.g., a crisis, project deadline—within the organization? While one would expect the content of these communications to be useful for event detection, this data is highly sensitive and often private. Instead, can events be discovered based only on structural information (i.e. message participants and their attributes)? For example, Romero et al. have shown that stock price shocks induce changes (e.g., higher clustering) in the structure of a hedge fund communication network.

Given the recent success of deep learning on graphs (Kipf and Welling 2016; Wu et al. 2019; Georgousis, Kenning, and Xie 2021) in node/graph classification, link prediction, and other tasks, it is natural to ask whether the same can also be useful for event detection. In particular, such an approach can combine techniques for graph classification and dynamic representation learning on graphs (Seo et al. 2018; Sankar et al. 2020; Guo et al. 2019). However, a key design question in this setting is whether to detect events based on the micro (node) or macro (graph) level dynamics. More specifically, the micro dynamics is captured via the application of a pooling operator to dynamic node embeddings (Nicolicioiu, Duta, and Leordeanu 2019; Pareja et al. 2019). For the macro dynamics, static snapshot embeddings are computed via pooling and their evolution is modeled via a recurrent architecture (e.g. an LSTM) (Seo et al. 2018; Zhao et al. 2019). Each of these approaches has implicit assumptions about the nature of events in the data.

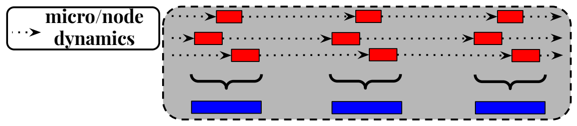
Figure 1 shows two event detection architectures, one based on micro and another based on macro dynamics. While they both apply a generic architecture shown in Figure 1a, they differ in the way dynamic representations for each graph snapshot are generated.

To illustrate the difference between micro and macro dynamics, let us revisit our organization example. Dynamic node embeddings are learned (non-linear) functions of the evolution of an employee's attributed neighborhood. These local embeddings are expected to be revealing of an employee's communication over time. Thus, (pooled) micro embeddings will capture average dynamic communication patterns within the organization. On the other hand, by pooling static node embeddings, we learn macro representations for the communication inside the organization at each timestamp. The recurrent architecture will then capture dynamic communication patterns at the organization level. Pooling and the RNN thus act as (spatial/temporal) functions that can be composed in different ways—e.g. f(g(x)) vs g(f(x))—each encoding specific inductive biases for event detection. We will show that the choice between micro and macro models has significant implications for event detection performance.

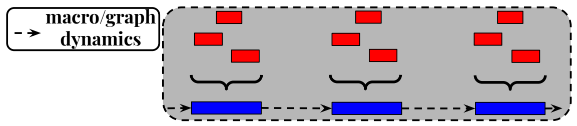
We propose DyGED (Dynamic Graph Event Detection), a graph neural network for event detection. DyGED combines a macro model with structural and temporal self-attention to account for application-specific node and time importances. To the best of our knowledge, our work is the first to apply either macro dynamics or self-attention for *the event detection task.* Despite its simplicity, differing from more recent



(a) Generic architecture for event detection



(b) Learning graph embeddings based on micro dynamics



(c) Learning graph embeddings based on macro dynamics

Figure 1: (a) Event detection on dynamic graphs based on a deep learning architecture. (b) At the micro scale, the dynamics is captured at node level using a temporal GNN architecture and then pooled for graph-level classification. (c) At the macro scale, the dynamics is captured at the graph level using an RNN over graph embeddings. Our work investigates how the dynamics at different scales affects event detection.

approaches based on micro dynamics, DyGED outperforms state-of-the-art solutions in three representative datasets.

One of the strengths of our study is its experimental evaluation. While the event detection problem has been studied by a recent paper (Deng, Rangwala, and Ning 2019)—based on micro dynamics—our work provides key insights into some of the challenges and possible strategies for effective event detection. This is partly due to our representative list of datasets covering mobility, communication, and user-generated content data. Our main contributions:

- We present the first study comparing micro and macro deep learning architectures for event detection on dynamic graphs, showing the importance of this design choice for event detection performance;
- We propose DyGED, a simple yet novel deep learning architecture for event detection based on macro dynamics. DyGED applies both structural and temporal self-attention to enable the effective learning of node and time dependent weights;
- We compare DyGED against several baselines—mostly based on a micro model—using three datasets. Our results show that DyGED outperforms the baselines by up to 8.5% while being scalable.

Problem Definition

Definition 1. ***Dynamic Graph:*** *A dynamic graph $\mathbb{G}$ is a sequence of T discrete snapshots $\langle G_1, G_2, \dots, G_T\rangle$ where G_t denotes the graph at timestamp t. G_t is a tuple (V, E_t, W_t, X_t) where V is a fixed set of n vertices, E_t is a set of m_t undirected edges, $W_t : E_t \to \mathbb{R}_+$ are edge weights, and $X_t : V \to \mathbb{R}^d$ gives d features for each node.*

In our earlier example regarding an organization, nodes in V represent organization members. An edge is created in E_t whenever associated members exchange a message during a time interval t and weights W_t might be numbers of messages exchanged. Finally, features X_t might include an individual's job position (static) and the total number of messages received by them during the time interval t (dynamic).

Definition 2. ***Event Label Function:*** *We define the function $\ell(G_{t-k:t}) \in \{0, 1\}$ to be an event labelling function of order k, where $G_{t-k:t} = \langle G_{t-k}, G_{t-k+1}, \dots, G_t\rangle$, and such that:*

$$\ell(G_{t-k:t}) = \begin{cases} 1, & \textit{if an event occurs at time } t \\ 0, & \textit{otherwise} \end{cases}$$

Events might also depend on the k previous snapshots $G_{t-k}, \dots, G_{t-1}$. This allows the function ℓ to model events that depend on how the graph changes. One can define a similar function ℓ_Δ for the early detection (or forecasting) of events Δ snapshots into the future.

Definition 3. ***Event Detection Problem:*** *Given a set of training instances $\mathbb{D}$, composed of pairs $(G_{t-k:t}, \ell(G_{t-k:t}))$, learn a function $\hat{\ell}$ that approximates the true ℓ for unseen snapshots.*

We treat event detection as a classification problem with two classes. To evaluate the quality of the learned function $\hat{\ell}$, we use AUC. In this paper, we propose $\hat{\ell}$ to be a neural network.

Proposed Model: DyGED[1]

We describe DyGED (Dynamic Graph Event Detection), a novel deep learning architecture for event detection on dynamic graphs. DyGED combines a Graph Convolutional Network and a Recurrent Neural Network to learn the macro dynamics correlated with labeled events. This backbone architecture is further enhanced by self-attention mechanisms in the structural and temporal domains.

We introduce notations A column-wise concatenation $[M_1, \dots, M_t] : \mathbb{R}^{n\times m_1} \times \dots \times \mathbb{R}^{n\times m_t} \to \mathbb{R}^{n\times(m_1+\ldots+m_t)}$ maps a sequence of matrices $M_1, \dots, M_t$ to a new matrix M such that $(M_t)_{i,j} = M_{i,\sum_{r=1}^{t-1} m_r + j}$. A row-wise concatenation as $[M_1; \dots; M_t] = [M_1^\intercal, \dots, M_t^\intercal]^\intercal$.

Main Components

Graph Convolutional Network GCNs are neural network architectures that support the learning of h-dimensional functions $\mathbf{GCN}(A, X) : \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times h}$ over vertices based on the graph adjacency matrix A and features X. For instance, a 2-layer GCN can be defined as follows:

$$\mathbf{GCN}(A, X) = \sigma\left(\hat{A}\,\sigma\left(\hat{A}XW^{(0)}\right)\,W^{(1)}\right)$$

[1]https://github.com/mertkosan/DyGED

where $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix with $\tilde{D}$ as weighted degree matrix and $\tilde{A} = I_n + A$ with I_n being an $n \times n$ identity matrix. $W^{(i)}$ is a trainable weight matrix for the i-th layer.

Pooling The output of the GCN described in the previous section is an embedding matrix Z_t for each graph snapshot G_t. In order to produce an embedding $\mathbf{z}_t$ for the entire snapshot, we apply a pooling operator v-Att$(Z_t) : \mathbb{R}^{n\times h} \to \mathbb{R}^h$. In particular, our model applies the self-attention graph pooling operator proposed in (Li et al. 2019):

$$\mathbf{z_t} = \text{v-Att}(Z_t) = \mathbf{softmax}(\mathbf{w}.\mathbf{tanh}(\Phi Z_t^T))Z_t$$

where $\Phi \in \mathbb{R}^{h\times h}$ and $w \in \mathbb{R}^h$ are learned attention weights.

Intuitively, v-Att re-weights the node embeddings enabling some nodes to play a larger role in the detection of events. In our experiments, we will show that these attention weights can be used to identify the most important nodes for our task.

Recurrent Neural Network We assume that events are correlated with the graph (i.e. macro) dynamics. Thus, our model applies an RNN to learn dynamic graph representations. More specifically, we give the pooled snapshot embeddings $\mathbf{z}_t$ as input to a standard $\mathbf{LSTM}(\mathbf{z'_{t-1}}, \mathbf{z_t}) : \mathbb{R}^h \times \mathbb{R}^h \to \mathbb{R}^h$ to produce dynamic graph representations $\mathbf{z'_t}$ based on a sequence of embeddings, instead of each node's (micro) dynamics.

Temporal Self-Attention The RNN enables our architecture to capture the dynamics via embeddings $\mathbf{z'_t}$. However, complex events might not be correlated only with the current graph representation but a window $Z'_t = [\mathbf{z'_{t-k}}; \ldots; \mathbf{z'_t}]$. For instance, in mobility-related events (e.g. sports games), changes in the mobility dynamics will arise a few hours before the event. Moreover, these correlations may vary within a dataset due to the characteristics of the type of event. We propose a self-attention operator t-Att$(Z'_t) : \mathbb{R}^{(k+1)\times h} \to \mathbb{R}^h$ for aggregating multiple dynamic embeddings:

$$\mathbf{z''_t} = \text{t-Att}(Z'_t) = \mathbf{softmax}(\mathbf{w'}.\mathbf{tanh}(\Phi' Z_t'^T))Z'_t$$

where $\Phi' \in \mathbb{R}^{h\times h}$ and $w' \in \mathbb{R}^h$ are learned attentions. Similar to v-Att, t-Att enables the adaptive aggregation of dynamic embeddings. To the best of our knowledge, we are the first to apply a similar self-attention mechanism—which might be of independent interest—in dynamic GNNs.

Classifier and Loss Function The final component of our model is an $MLP(z''_t) : \mathbb{R}^h \to \mathbb{R}^2$ that returns (nonlinear) scores for each possible outcome (i.e., event/no event) Y_t. We use *cross-entropy* as our loss function. Note that event detection is a highly imbalanced problem—i.e. events are rare. We address this challenge by weighting our loss function terms with class ratios. As a result, false negatives are more penalized than false positives.

$$-\sum_{t=k+1}^{T}(1-x)\ell(G_{t-k:t})\log(Y_{t,1}) + x(1-\ell(G_{t-k:t}))\log(Y_{t,2})$$

where ℓ is the event label from Definition 2. Moreover, x positive (i.e., events) sample ratio in the training set.

Algorithm 1: DyGED Forward Algorithm

```
Require: Sequence of snapshots $G_{t-k:t}$, previous dynamic state
    $z'_{t-k-1}$
Ensure: Event probability
1: for $\tau \in \{t-k, \ldots, t\}$ do
2:     $Z_\tau \leftarrow GCN(G_\tau, X_\tau)$
3:     $z_\tau \leftarrow$ v-Att$(Z_\tau)$
4:     $z'_\tau \leftarrow LSTM(z'_{\tau-1}, z_\tau)$
5: end for
6: $z''_t \leftarrow$ t-Att$([z'_{t-k}; \ldots; z'_t])$
7: return $MLP(z''_t)$
```

DyGED and its Variants

Algorithm 1 provides an overview of the forward steps of DyGED. It receives a sequence of snapshots $G_{t-k:t}$ and the previous dynamic (LSTM) state z'_{t-k-1} as inputs. The output is the event probability for G_t. Steps 3 and 6 correspond to structural and temporal self-attention, respectively. In order to evaluate some of the key decisions involved in the design of DyGED, we also consider the following variations of our model:

- DyGED-CT (with concatenation): Replaces the LSTM (step 4) and t-Att (step 6) operators by a concatenation, with $z''_t = ([z_{t-k}, \ldots, z_t])$.
- DyGED-NL (no LSTM): Removes the LSTM operator (step 4) from Algorithm 1, with z''_t = t-Att$([z_{t-k}; \ldots; z_t])$.
- DyGED-NA (no attention): Removes the temporal self-attention operator t-Att (step 5), with $z''_t = z'_t$.

Experiments

Datasets

Table 2 shows the main statistics of our datasets. The snapshot period is the interval $[time_t, time_t + \Delta p)$ covered by each snapshot G_t, where Δp denotes the period. We use representative datasets of relevant event detection applications. NYC Cab[2] is an example of a mobility network with geotagged mass-gathering events (e.g., concerts, protests). We use baseball games involving the Yankees or the Mets are the events of interest. Hedge Fund (Romero, Uzzi, and Kleinberg 2016) is a communication network for decision making in high-risk environments—as in other business settings and emergency response. Each snapshot covers activities in a day, and events are price shocks—unexpected changes—in the S&P500. Twitter Weather relates user-generated content with extreme events (e.g., terrorist attacks, earthquakes). Weather events—with monetary damage of at least $50M—were collected from the US National Climatic Data Center records.[3] We also created a larger version of Twitter Weather with 1000 words.

Experimental Settings

Baselines: We consider recent approaches that either focus on micro (node-level) dynamics or are designed for graph classification. If the pooling is necessary, we apply our v-Att module.

[2]https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[3]https://www.ncdc.noaa.gov/stormevents/

	Method	NYC Cab	Hedge Fund	TW	TW Large
Baselines: Micro Dynamics	EvolveGCN	$0.842^{**} \pm 0.008$	$0.718^{**} \pm 0.011$	$0.782^{**} \pm 0.012$	$0.731^{**} \pm 0.011$
	ASTGCN	$0.903^{**} \pm 0.003$	$0.753^{*} \pm 0.022$	$0.747^{**} \pm 0.018$	$0.722^{**} \pm 0.014$
	DynGCN	$0.901^{**} \pm 0.003$	$0.679^{**} \pm 0.030$	$0.713^{**} \pm 0.012$	$0.709^{**} \pm 0.007$
Classification	DiffPool	$0.887^{**} \pm 0.003$	$0.690^{**} \pm 0.020$	$0.766^{**} \pm 0.014$	$0.728^{**} \pm 0.007$
Proposed: Macro Dynamics	*DyGED-CT*	$\underline{0.910} \pm 0.009$	0.776 ± 0.012	$0.775^{**} \pm 0.014$	$0.743^{**} \pm 0.014$
	DyGED-NL	$\mathbf{0.912} \pm 0.004$	$0.779^{*} \pm 0.012$	$0.791^{**} \pm 0.014$	$\underline{0.752}^{*} \pm 0.020$
	DyGED-NA	$0.896^{**} \pm 0.004$	$\underline{0.784}^{*} \pm 0.014$	$\underline{0.800}^{*} \pm 0.009$	$0.734^{**} \pm 0.012$
	DyGED	$0.905^{*} \pm 0.004$	$\mathbf{0.787} \pm 0.015$	$\mathbf{0.810} \pm 0.012$	$\mathbf{0.760} \pm 0.014$

Table 1: AUC scores of event detection methods. The highest and second highest values for each column are in bold and underlined, respectively. Our methods, accounting for macro dynamics, achieve the best results, outperforming the best baseline (ASTGCN) by 4.5% on average and up to 8.5% (on Twitter Weather). We performed a paired t-test comparing the best model against the others (markers ** and * indicate p-value $< .01$ and $< .05$, respectively).

	NYC Cab	Hedge Fund	TW	TW-Large
#Nodes (avg)	263	330	300	1000
#Edges (avg)	3717	557	1142	10312
#features	6	5	300	300
#Snapshots	4464	690	2557	2557
Snap. Period	hour	day	day	day
#Events	162	55	287	287

Table 2: The statistics of the datasets. TW is Twitter Weather.

- **DynGCN (Deng, Rangwala, and Ning 2019):** State-of-the-art for event detection that combines representations from a GCN at each snapshot with a temporal encoder.
- **EvolveGCN (Pareja et al. 2019):** Combines recurrent and GCNs to generate dynamic node embeddings.
- **ASTGCN (Guo et al. 2019):** It combines spatial and temporal-attention mechanisms. We consider k previous time dependencies instead of daily, weekly, and monthly ones.
- **DiffPool (Ying et al. 2018):** Computes graph embeddings via a differentiable graph pooling method. Because this model is designed for classification, it does not account for the dynamics.

Other Settings We use p-fold nested cross-validation, where p was set based on event frequency. Each method runs 20 times per train/test split. We find that training using Adam optimization with learning rate, dropout rate, and the batch size set to 0.005, 0.2, and 100, respectively. The embedding size is set to 64.

Event Detection Accuracy

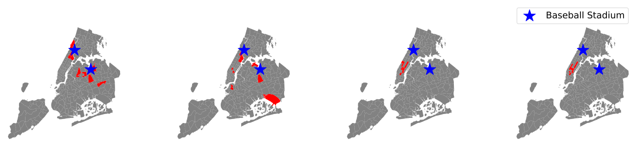
Table 1 shows the event detection accuracy results. For the approaches that consider a sliding window, reported results are the best ones among window sizes $(k + 1)$ varying from one to five. The optimal window size for all these methods is either four or five. Results show that DyGED outperforms the competing approaches in all datasets. In particular, DyGED outperforms ASTGCN—best baseline—by 0.2%, 4.5%, 8.5%, and 5.2% for NYC Cab, Hedge Fund, Twitter Weather, and Twitter Weather Large, respectively (4.5% on average). Notice that, different from most baselines, our approach captures the macro-dynamics correlated with events. DyGED-NL and DyGED, which adopt temporal self-attention, achieve the best results indicating that it enables the learning of adaptive weights for different snapshots. Moreover, DyGED-NA and DyGED—using recurrent neural network to capture the macro dynamics—achieve better performance for the Hedge Fund and Twitter Weather datasets.

	NYC Cab	Hedge Fund	TW	TW-Large
EvolveGCN	1.912	0.303	0.949	4.65
ASTGCN	13.57	2.131	6.916	24.41
DynGCN	0.064	0.019	0.080	0.622
DiffPool	0.057	0.016	0.045	0.482
DyGED	0.066	0.017	0.048	0.479

Table 3: Testing times (in secs.) for all methods and datasets.

Event Detection Efficiency

Table 3 shows testing times (in secs.) for a batch size of 100 data points for NYC Cab, Hedge Fund, Twitter Weather, and Twitter Weather Large, respectively. The window size is set to 4. Results show that DyGED is scalable—up to 206 and 29 times faster than the top 2 baselines (ASTGCN and EvolveGCN, respectively).



(a) DyGED Attention (b) Betweenness (c) Clustering (d) Degree

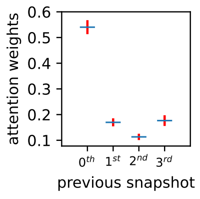
Figure 2: Top 10 taxi zones (in red) based on various metrics for NYC Cab dataset. DyGED Attention mechanism finds closer taxi zones to the stadiums.

Importance via Attention

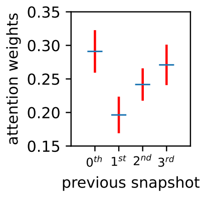
DyGED applies node and time self-attention for event detection. Here, we analyze these attention weights as a proxy to infer node and time *importance*. We notice that the use of attention weights for interpretability is a contentious topic (Jain and Wallace 2019; Wiegreffe and Pinter 2019). We still find that these learned weights provide interesting insights regarding the role played by self-attention in our model.

Node Attention A critical task in event detection on graphs is to measure the importance of nodes and subgraphs (Ying et al. 2019). To answer this question, we analyze attention weights learned the v-Att(.) operator—normalized by the softmax function. Figure 2 shows the top 10 taxi zones based on each importance measure for the NYC Cab dataset. Our attention weights find taxi zones near the baseball stadiums, whereas topology-based baseline measures select stations in downtown Manhattan and the airports. In Twitter Weather, our solution set contains "fire", "tree", and "snow" as the top words while the topology-based baselines have "weather", "update", and "barometer". The words found using our measure are more strongly associated with events of interest.

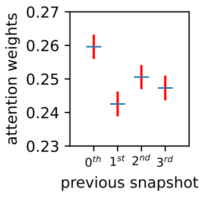
Time (Snapshot) Attention We also propose a time importance module that uses temporal self-attention weights via the function t-Att(.), to measure how the past snapshots (time) affect event detection. We use three previous (i.e., $k = 4$) and the current snapshot in our experiments. Figure 3 shows the attention weights (output of softmax) for snapshots (with mean and standard deviation). For NYC Cab, the current snapshot has significantly higher weights. However, the remaining datasets reveal more interesting attention patterns. For instance, in Hedge Fund, the importance of earlier weights can be associated with the definition of an event—a stock market shock. For Twitter Weather, events often last few days, and thus weights are expected to be more uniform.



(a) NYC Cab



(b) Hedge Fund



(c) Twitter Weather

Figure 3: Illustration of the normalized attention weights of current (0^{th}) and previous three snapshots for all datasets. Results show the history plays a role in predicting the event.

Conclusions

This paper is focused on event detection on dynamic graphs. We have proposed a deep learning based method, DyGED, which learns correlations between the graph macro dynamics—i.e. a sequence of temporal graph representations—and events. We compared DyGED against multiple baselines using a representative set of datasets. Our approach outperformed the baselines in terms of accuracy while being scalable. We also showed how our method can be applied to provide interpretability via self-attention on nodes and snapshots.

Acknowledgments

This work is funded by NSF via grant IIS 1817046. Some material is based upon work supported by the Air Force Office of Scientific Research under Minerva award number FA9550-19-1-0354.

References

Deng, S.; Rangwala, H.; and Ning, Y. 2019. Learning Dynamic Context Graphs for Predicting Social Events. In *SIGKDD*.

Georgousis, S.; Kenning, M. P.; and Xie, X. 2021. Graph deep learning: State of the art and challenges. *IEEE Access*.

Guo, S.; Lin, Y.; Feng, N.; Song, C.; and Wan, H. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *AAAI*.

Jain, S.; and Wallace, B. C. 2019. Attention is not explanation. *arXiv preprint arXiv:1902.10186*.

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

Li, J.; Rong, Y.; Cheng, H.; Meng, H.; Huang, W.; and Huang, J. 2019. Semi-supervised graph classification: A hierarchical graph perspective. In *The Web Conference*, 972–982.

Li, Z.; Sun, D.; Zhu, R.; and Lin, Z. 2017. Detecting event-related changes in organizational networks using optimized neural network models. *PloS one*.

Nicolicioiu, A.; Duta, I.; and Leordeanu, M. 2019. Recurrent Space-time Graph Neural Networks. volume 32, 12838–12850.

Pareja, A.; Domeniconi, G.; Chen, J.; Ma, T.; Suzumura, T.; Kanezashi, H.; Kaler, T.; and Leisersen, C. E. 2019. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. *arXiv preprint arXiv:1902.10191*.

Romero, D. M.; Uzzi, B.; and Kleinberg, J. 2016. Social Networks Under Stress. In *WWW*.

Sankar, A.; Wu, Y.; Gou, L.; Zhang, W.; and Yang, H. 2020. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. In *WSDM*.

Seo, Y.; Defferrard, M.; Vandergheynst, P.; and Bresson, X. 2018. Structured sequence modeling with graph convolutional recurrent networks. In *NeurIPS*. Springer.

Wiegreffe, S.; and Pinter, Y. 2019. Attention is not not explanation. *arXiv preprint arXiv:1908.04626*.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

Ying, R.; Bourgeois, D.; You, J.; Zitnik, M.; and Leskovec, J. 2019. Gnnexplainer: Generating explanations for graph neural networks. In *NeurIPS*.

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

Zhao, L.; Song, Y.; Zhang, C.; Liu, Y.; Wang, P.; Lin, T.; Deng, M.; and Li, H. 2019. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE ITS*.