Event Detection on Dynamic Graphs[*]

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 a representative set of datasets, demonstrates that DyGED outperforms competing solutions in terms of event detection accuracy by up to 8.5% while being more scalable than the top alternatives. We also present case studies illustrating key features of our model.

1 Introduction

Event detection on dynamic graphs is a relevant task for effective decision-making in many organizations (Li et al. 2017; Leetaru and Schrodt 2013). In graphs, entities and their interactions are represented as (possibly attributed) 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.

---

[*]Long version of *Graph Macro Dynamics with Self-Attention for Event Detection* published in DLG-AAAI'23.

(Romero, Uzzi, and Kleinberg 2016) have shown that stock price shocks induce changes (e.g., higher clustering) in the structure of a hedge fund communication network. Shortly after the 2011 earthquake and tsunami, Japanese speakers expanded their network of communication on Twitter (Lu and Brelsford 2014).

Given the recent success of deep learning on graphs (Kipf and Welling 2016; Hamilton, Ying, and Leskovec 2017; 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 (Zhang et al. 2018; Errica et al. 2020) and dynamic representation learning on graphs (Nicolicioiu, Duta, and Leordeanu 2019; Seo et al. 2018; Pareja et al. 2019; Zhao 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. For simplicity, we will assume that the pooling operator is the average. 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) func-

tions 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.

This paper investigates the event detection problem on dynamic graphs. 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 approaches based on micro dynamics, DyGED outperforms state-of-the-art solutions in three representative datasets. These findings also have implications for other graph-level analytics tasks on dynamic graphs, such as anomaly detection, regression, and prediction.

One of the strengths of our study is its extensive 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. Moreover, we present a few case studies illustrating the key features of our approach. Our main contributions are as follows:

- 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. We also provide case studies illustrating relevant features of our model (e.g. how its embeddings can be used for diagnosis).

2 Related Work

Event detection on graphs: There is a diverse body of work on event detection using graphs and other types of structured data (Sakaki, Okazaki, and Matsuo 2010; Ramakrishnan et al. 2014; Atefeh and Khreich 2015) in the literature. Moreover, other popular tasks such as anomaly (Ranshous et al. 2015; Rayana and Akoglu 2015), change point (Akoglu and Faloutsos 2010; Peel and Clauset 2015), and intrusion (Staniford-Chen et al. 1996) detection are related to unsupervised event detection (Chen and Neill 2014; Rayana and Akoglu 2015; Rozenshtein et al. 2014). Here, we focus on a supervised version of the problem where graph snapshots are

(a) Generic architecture for event detection

(b) Learning graph embeddings based on micro dynamics

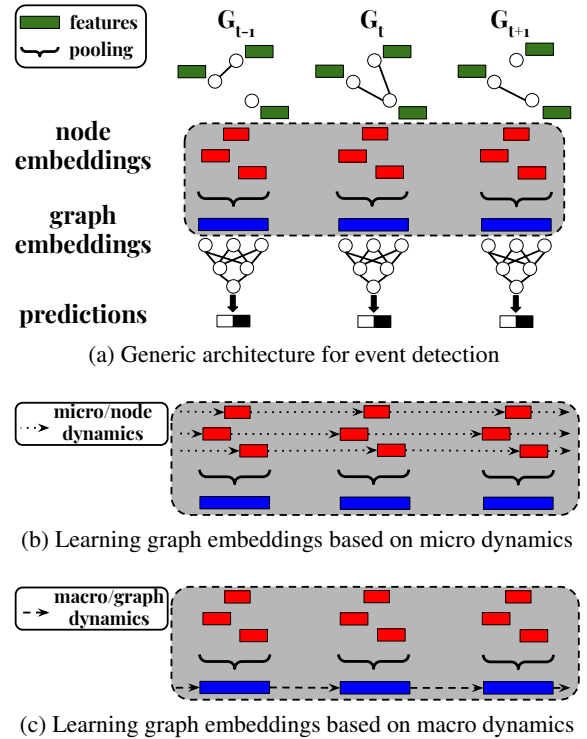(c) Learning graph embeddings based on macro dynamics

Figure 1: (a) Event detection on dynamic graphs based on a generic 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 pooled (static) GNN node embeddings. Our work investigates how the dynamics at different scales affects event detection performance.

labeled depending on whether an event happened within the snapshot's time window. We assume that events are defined based on an external source of information, not being fully identifiable from—but still being correlated with—the observed data (Li et al. 2017; Deng, Rangwala, and Ning 2019). Some studies distinguish the event detection from the event forecasting problem, as the second also allows for predictions of events far into the future (Ramakrishnan et al. 2014). However, such a distinction is less relevant when events are external to the data (Deng, Rangwala, and Ning 2019). Recently in (Hu et al. 2021), the authors apply a dynamic graph to identify events on time series. We focus on the detection of events with graphs as inputs.

The classical approach for event detection, including in the case of graphs, is to rely on features designed by experts (Aggarwal and Subbian 2012; Ning et al. 2018). A framework for automatically combining multiple social network metrics, such as modularity and clustering coefficient, as features for terrorist activity detection using a neural network is introduced in (Li et al. 2017). However, these metrics, which are based on expert knowledge, may not generalize to many applications, and they are potentially non-exhaustive—i.e. other relevant metrics might be missing.

Graph kernels: In machine learning, kernel methods, such as Support Vector Machines, have motivated a long

literature on graph kernels (Kriege, Johansson, and Morris 2020). A kernel is a function that computes the similarity between two objects in a particular application and thus their design often requires expert-knowledge. In the case of graph kernels, graphs are compared based on their substructures—e.g., node neighborhoods (Kriege and Mutzel 2012), graphlets (Shervashidze et al. 2009), random walks (Kashima, Tsuda, and Inokuchi 2003). However, recent studies have shown that such features are often outperformed by those learned directly from data (Zhang et al. 2018).

Temporal node embeddings: Modeling temporal evolution of nodes based on their connections over time is a well-studied problem in the literature. The temporal information can be encoded using temporal point process (Trivedi et al. 2019), random walks on temporal edge orderings (Nguyen et al. 2018) or a joint optimization of temporal node embeddings (Singer, Guy, and Radinsky 2019). While these methods focus on micro embeddings, (Lu et al. 2019) proposes taking into account macro information by incorporating the number of modified edges in the graph. However, we notice that (Lu et al. 2019) does not apply to our setting since it does not find node embeddings for every timestamp.

Graph neural networks (GNNs) for dynamic graphs: Deep learning on graphs is an effort to reproduce the success achieved by deep neural networks (e.g. CNNs, RNNs) on graphs (Gori, Monfardini, and Scarselli 2005; Duvenaud et al. 2015). A key advance was the introduction of Graph Convolutional Networks (GCNs) (Kipf and Welling 2016), which outperform traditional approaches for semi-supervised learning. Later, GraphSage (Hamilton, Ying, and Leskovec 2017) and Graph Attention Networks (GATs) (Veličković et al. 2017) were proposed to increase the scalability and exploit the attention mechanism in GNNs, respectively.

Following the work listed above, there has been an outburst of extensions of GNNs for dynamic graphs (Skarding, Gabrys, and Musial 2020). One approach is to apply standard GNNs to a static graph where multiple snapshots are represented as a single multi-layered graph (Yan, Xiong, and Lin 2018; Guo et al. 2019) with additional temporal edges. More recently, several studies have combined recursive architectures, such as LSTMs, with GNNs. This can be achieved via macro models, which stack graph-level GNN representations as a sequential process (Taheri, Gimpel, and Berger-Wolf 2019; Seo et al. 2018). Another alternative are micro models, which apply the sequential process at the node level to generate dynamic node embeddings (Sankar et al. 2020; Pareja et al. 2019; Guo et al. 2019).

Dynamic GNNs have been applied mostly for local tasks (e.g. node classification). On the other hand, the most popular graph-level task for GNNs is graph classification (Niepert, Ahmed, and Kutzkov 2016; Lee, Rossi, and Kong 2018; Ying et al. 2018), which assumes the input to be static. Our paper is focused on event detection on dynamic graphs, a challenging graph-level task also studied in (Deng, Rangwala, and Ning 2019)—one of our baselines. Different from their work, DyGED applies self-attention at node and time domains to capture the macro dynamics correlated with events. Self-attention was also recently applied in (Li et al. 2019) and (Sankar et al. 2020), which were focused on graph clas-

sification and link prediction, respectively. We show that DyGED equipped with self-attention outperforms state-of-the-art event detection methods.

3 Problem Definition

Supervised event detection on dynamic graphs consists of learning how to detect events based on a few recent graph snapshots using training data (i.e. labeled events).

Definition 1. *Dynamic Graph: A dynamic graph $\mathbb{G}$ is a sequence of T discrete snapshots $\langle G_1, G_2, \ldots, G_T \rangle$ 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 (Section 1), nodes in V represent members of the organization. An edge is created in E_t whenever their 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 $G_{t-k:t} = \langle G_{t-k}, G_{t-k+1}, \ldots, G_t \rangle$, and such that:*

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

Events might depend not only on the current graph snapshot G_t but also on the k previous snapshots $G_{t-k}, \ldots, G_{t-1}$. 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 apply a traditional evaluation metric (AUC (Fawcett 2006)) from supervised learning. In this paper, we propose $\hat{\ell}$ to be a neural network.

4 Proposed Model: DyGED

We describe DyGED (Dynamic Graph Event Detection), a simple yet 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 (i.e. graph-level) dynamics correlated with labeled events. This backbone architecture is further enhanced by self-attention mechanisms in the structural and temporal domains. First, we introduce the main components of our architecture. Next, we describe DyGED and some of its variations.

We introduce notations for a few basic operations to describe our architectures in a compact form. A column-wise concatenation $[M_1, \ldots, M_t]$: $\mathbb{R}^{n \times m_1} \times \ldots \times$

$\mathbb{R}^{n \times m_t} \rightarrow \mathbb{R}^{n \times (m_1+...+m_t)}$ maps a sequence of matrices $M_1, \ldots, M_t$ to a new matrix M such that $(M_t)_{i,j} = M_{i,\sum_{r=1}^{t-1} m_r + j}$. Similarly, we denote a row-wise concatenation as $[M_1; \ldots; M_t] = [M_1^\mathsf{T}, \ldots, M_t^\mathsf{T}]^\mathsf{T}$.

4.1 Main Components

We describe the main components of our neural network architecture for event detection on dynamic graphs (DyGED).

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

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

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 weight matrix for the i-th layer to be learned during training, with $W^{(1)} \in \mathbb{R}^{d \times h'}$, $W^{(2)} \in \mathbb{R}^{h' \times h}$, and h (h') being the embedding size in the output (hidden) layer. Moreover, $\sigma(.)$ is a non-linear activation function such as **ReLU**.

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 $\text{v-Att}(Z_t) : \mathbb{R}^{n \times h} \rightarrow \mathbb{R}^h$. 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 Long Short-Term Memory $\mathbf{LSTM}(\mathbf{z'_{t-1}}, \mathbf{z_t}) : \mathbb{R}^h \times \mathbb{R}^h \rightarrow \mathbb{R}^h$ to produce dynamic graph representations:

$$\mathbf{z'_t} = \mathbf{LSTM}(\mathbf{z'_{t-1}}, \mathbf{z_t})$$

Notice that $\mathbf{z'_t}$ is based on a sequence of static embeddings, instead of each node's (micro) dynamics. In our experiments, we will compare these two approaches using a diverse collection of datasets.

Temporal Self-Attention The RNN component described in the previous section enables our architecture to capture the graph dynamics via embeddings $\mathbf{z'}$. 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),

---

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:     $z_\tau \leftarrow \text{v-Att}(Z_\tau)$
4:     $z'_\tau \leftarrow LSTM(z'_{\tau-1}, z_\tau)$
5: **end for**
6: $z''_t \leftarrow \text{t-Att}([z'_{t-k}; \ldots; z'_t])$
7: **return** $MLP(z''_t)$

---

changes in the mobility dynamics will arise a few hours before the event takes place. Moreover, these correlations might vary within a dataset due to the characteristics of each type of event. Thus, we propose a self-attention operator $\text{t-Att}(Z'_t) : \mathbb{R}^{(k+1) \times h} \rightarrow \mathbb{R}^h$ 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 attention weights.

Similar to v-Att (Section 4.1), 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 GNN architectures.

Classifier and Loss Function The final component our model is a Multi-Layer Perceptron $MLP(z''_t) : \mathbb{R}^h \rightarrow \mathbb{R}^2$ that returns (nonlinear) scores for each possible outcome (i.e., event/no event) Y_t. Given training data with event labels $\ell(G_{t-k:t})$, the parameters of our model are learned (end-to-end) by minimizing the *cross-entropy* of event predictions $Y = \{Y_{k+1}, \ldots, Y_T\}$. 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 (Ho and Wookey 2020). 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 and $1-x$ positive (i.e., events) and negative sample ratios in the training set, respectively.

4.2 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. Notice that our assumption that macro dynamics of the graph is correlated with events of interest leads to a simple and modular model. Steps 3 and 5 correspond to the 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 contatenation): Replaces the LSTM (step 4) and t-Att (step 5) 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 $z_t'' = \text{t-Att}([z_{t-k}; \ldots; z_t])$.

- DyGED-NA (no attention): Removes the temporal self-attention operator t-Att (step 5), with $z_t'' = z_t'$.

Time Complexity Table 1 shows the time complexities for different variations of DyGED discussed in this section. Our methods are scalable as the time complexities are linear with the number of vertices (n) and edges (m) in the input graph.

	GCN + Pooling	LSTM	t-Att	MLP
DyGED-CT	$O((mh + nh^2)l_1)$	-	-	$O(kh^2 + h^2 l_2)$
DyGED-NL	$O((mh + nh^2)l_1)$	-	$O(kh^2)$	$O(h^2 l_2)$
DyGED-NA	$O((mh + nh^2)l_1)$	$O(h^2)$	-	$O(h^2 l_2)$
DyGED	$O((mh + nh^2)l_1)$	$O(h^2)$	$O(kh^2)$	$O(h^2 l_2)$

Table 1: Time complexities of our methods: m, n, h, l_1, k, l_2 denote numbers of edges, nodes, embedding dimension, layers in GCN, past snapshots, layers in MLP. We assume the initial node feature dimension for GCN, input, cell, and output dimension for LSTM equal to embedding dimension h.

5 Experiments

We evaluate DyGED—which is our approach for event detection on dynamic graphs—and its variations using a diverse set of datasets. We compare our solutions against state-of-the-art baselines for event detection, graph classification, and dynamic GNNs in terms of accuracy (Sec. 5.3) and efficiency (Sec. 5.7). We also provide a visualization of prediction scores to give more insight into the results (Sec. 5.4. Furthermore, we present more studies to have a more detailed picture of the effectiveness of DyGED across representative application scenarios using event embeddings (Sec. 5.6), importance via attention (Sec. 5.5), and ablation study including feature and pooling operator variations (Sec. 5.8). The implementation of DyGED and datasets are available online[1].

	NYC Cab	Hedge Fund	TW	TW-Large
#Nodes (avg)	263	330	300	1000
#Edges (avg)	3717	557	1142	10312
#static features	6	5	300	300
#dynamic features	3	3	3	3
#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.

---

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

5.1 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. These datasets are representatives of relevant event detection applications. NYC Cab is an example of a mobility network with geo-tagged mass-gathering events (e.g., concerts, protests). Hedge Fund is a communication network for decision making in high-risk environments—as in other business settings and emergency response. Twitter Weather relates user-generated content with extreme events (e.g., terrorist attacks, earthquakes).

NYC Cab: Dataset based on sports events and hourly numbers of passengers transported between cab zones in NYC.[2] Nodes, edges, and their weights represent cab zones, inter-zone trips, and numbers of passengers transferred, respectively. Static node features are lat-long coordinates, boroughs, lengths, areas, and service zones. Dynamic features are the node degree, betweenness centrality, and clustering coefficient within a snapshot (i.e. one hour period). Baseball games involving the Yankees or the Mets in NYC are the events of interest.

Hedge Fund (Romero, Uzzi, and Kleinberg 2016): Dynamic network of employees at a hedge fund and their communications. Stock market shocks between January 2009 and September 2011 are the events. Nodes, edges, and edge weights represent employees, communications, and the total number of messages exchanged, respectively. Node features are employee's personal information such as company name, hiring time, gender, and position. We consider the node degree, betweenness centrality, and clustering coefficient as dynamic features. Each snapshot covers activities in a day, and events are price shocks—unexpected changes (Romero, Uzzi, and Kleinberg 2016)— in the S&P500.

Twitter Weather: Dataset integrating weather-related tweets and significant weather events in the US from 2012 to 2018. Tweets were extracted from a large corpus made available by the Internet Archive[3]. Nodes, edges, and edge weights represent English words, word co-occurrences, and the number of co-occurrences, respectively. Starting from a small set of weather-related words, we employed an existing algorithm (Wang et al. 2016) to expand the set to 300 words. Pre-trained word2vec[4] (Mikolov et al. 2013) vectors were applied as static node features, whereas dynamic features are the node degree, betweenness centrality, and clustering coefficient during a one day interval. Weather events—with monetary damage of at least $50M—were collected from the US National Climatic Data Center records.[5] We also created a larger version of Twitter Weather with 1000 words.

5.2 Experimental Settings

Baselines: We consider recent approaches that either focus on micro (node-level) dynamics (Deng, Rangwala, and Ning 2019; Pareja et al. 2019; Guo et al. 2019) or are designed

---

[2] https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
[3] https://archive.org/details/twitterstream
[4] https://code.google.com/archive/p/word2vec/
[5] 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 3: 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).

for graph classification (Ying et al. 2018). If it is necessary, we apply our v-Att module (see Section 4.1) to get graph embeddings from node embeddings.

- **DynGCN (Deng, Rangwala, and Ning 2019):** State-of-the-art architecture for event detection that combines representations from a GCN at each snapshot with a temporal encoder that carries information from the past.

- **EvolveGCN (Pareja et al. 2019):** Combines recurrent and graph convolutional neural networks to generate dynamic node embeddings.

- **ASTGCN (Guo et al. 2019):** Graph convolutional network originally proposed for traffic prediction. It combines spatial and temporal-attention mechanisms. We adapt ASTGCN to our problem setting by considering 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   Train/Test Splits: We evaluate the methods using p-fold nested cross-validation (Bergmeir and Benítez 2012), where p was set based on event frequency. Each method runs 20 times per train/test split (at least 100 repetitions/method), and we report average results. Train/test splits of $3720/744$, $575/115$, and $2192/365$ snapshots are applied for the NYC Cab, Hedge Fund, Twitter Weather datasets, respectively.

Ratio of positive and negative samples: Overall percentages of positive samples (events) are 3.62%, 7.97%, and 11.22% for the NYC Cab, Hedge Fund, and Twitter Weather dataset, respectively. As we use nested cross-validation, train/test event ratios vary (depending on the fold) from/to 3.46/4.18%-3.61/3.37%, 3.57/12.5%-8.37/7.14% and 11.17/18.23%-13.53/4.14% for NYC Cab, Hedge Fund, and Twitter Weather respectively.

Hyper-parameters: We tune the hyper-parameters of our methods and baselines with a grid search. 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, works well for our methods. The number of GCN layers,

MLP layers, the size of the embedding set to 2, 2, and 64, respectively, are good choices for model parameters.

Quality metric: We compare the quality of the predictions by the methods using the Area under the ROC curve (AUC) (Fawcett 2006).

Hardware: We run our experiments on a machine with NVIDIA GeForce RTX 2080 GPU (8GB of RAM) and 32 Intel Xeon CPUs (2.10GHz and 128GB of RAM).

5.3   Event Detection Accuracy

Table 3 shows the event detection accuracy results in terms of AUC. 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. We use only static node features in our main experiments, and the effect of dynamic features will be shown in Section 5.8.

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/times. 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.

5.4   Visualization of Event Scores

Besides evaluating event detection approaches in terms of accuracy, it is also important to analyze how event scores are related to true events in the data. This is particularly relevant when events are determined based on a threshold.

Figure 2 shows the prediction scores for a set of events from the Hedge Fund dataset. It illustrates how events are distributed over time and also how specific events are predicted by the methods. Events are marked as vertical (red) lines. Moreover, scores are (min-max) scaled versions of the event (yes) class score (i.e. $Y_{t,1}$, as defined in Section 4.1).
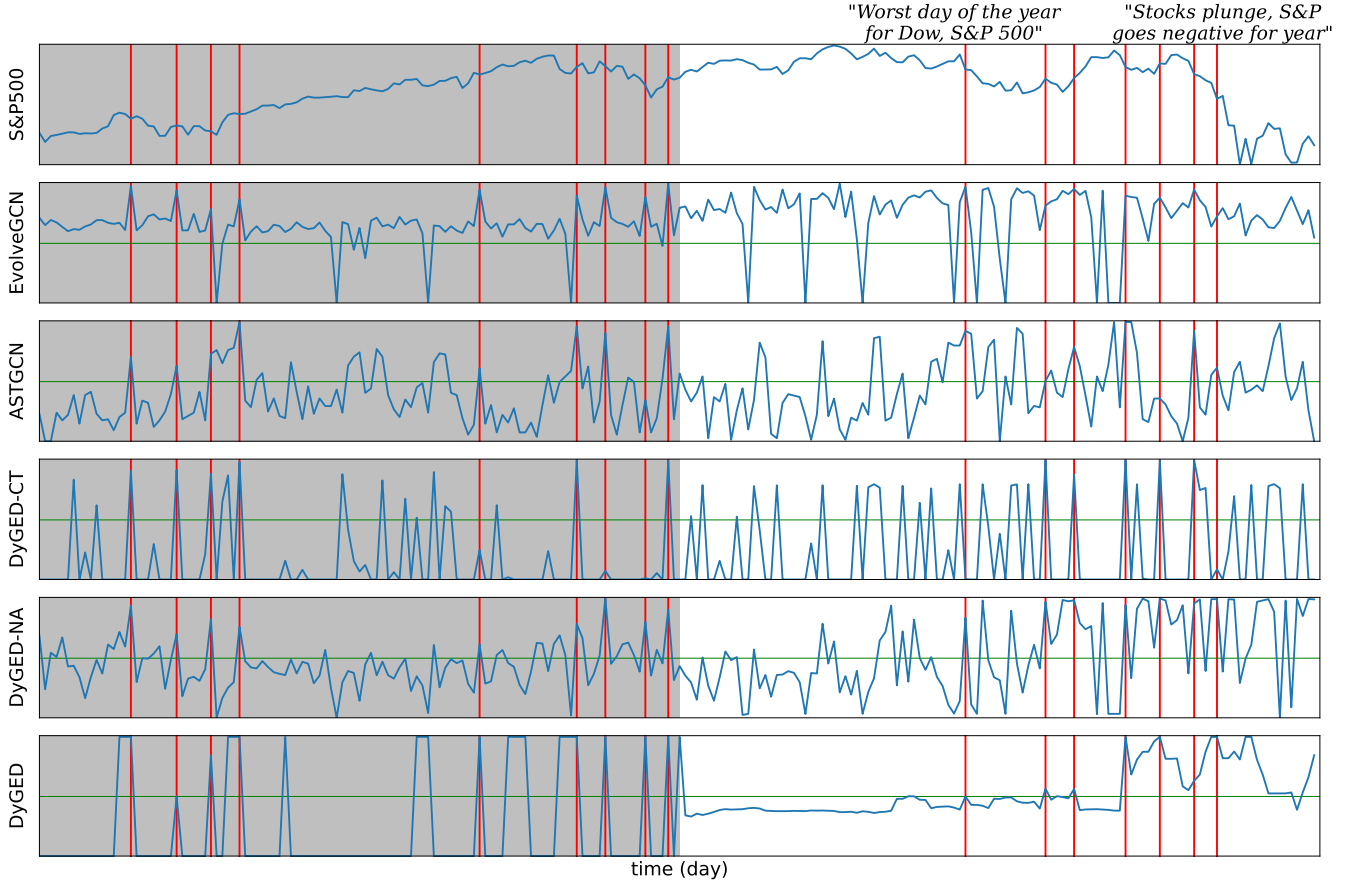
Figure 2: Scaled prediction scores ($[0, 1]$) given by some of the event detection methods for Hedge Fund. Events are marked as vertical (red) lines. The first (grey) half covers some of the training events, while the remaining (white) is in the test set. Some events are annotated for illustration. Hedge Fund annotations are headlines from CNBC on the day of the event. Our method DyGED can identify several of the events while keeping the false positive rate lower.

We show events and scores for part of the training window (in gray) and also the complete test window (in white) of a particular fold. We also show the value of the S&P500 index and some headlines of the day from the business section of the CNBC News website.[6] These visualizations provide important insights regarding the nature of events in these datasets and also about the performance of event detection schemes.

Notice that prediction scores are correlated with events across methods, particularly the top-performing ones. However, most methods tend to suffer from false positive errors. This shows that the relationship between the graph dynamics and events is often weak (or noisy), which is a challenge for event detection methods.

Prediction scores also give us a better understanding of the differences between our method (DyGED) and baselines EvolveGCN and ASTGCN. Different from DyGED, these baselines capture the micro (or node-level) dynamics of the graph. On the other hand, our approach focuses on the macro (or graph-level) dynamics. First, DyGED is effective at fitting the training data. Moreover, DyGED also achieves better performance during testing, producing significantly fewer

false positives. These results are consistent with the quantitative analysis provided in Section 5.3.

5.5 Importance via Attention

DyGED applies node and time self-attention weights for event detection. Here, we analyze these attention weights as a proxy to infer *importance* node and time importance. We notice that the use of attention weights for interpretability is a contentious topic in the literature (Jain and Wallace 2019; Pruthi et al. 2019; Wiegreffe and Pinter 2019). Still, we find that these learned weights to be meaningful for our datasets. They also 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) based on the events of interest. As a step towards answering this question, we analyze attention weights learned the v-Att(.) operator—normalized by the softmax function. For each node, we compute the average weight learned. For a comparison, we also consider the following classical node importance measures from the network science literature: *degree, betweenness centrality*, and *clustering coefficient*.

---

[6]https://www.cnbc.com

(a) DyGED Attention Weights   (b) Betweenness   (c) Clustering   (d) Degree
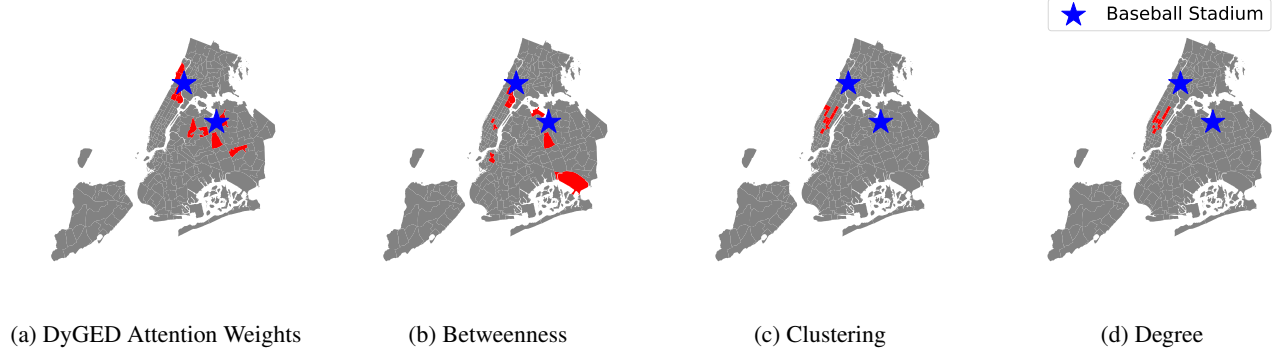
Figure 3: Top 10 taxi zones (in red) based on various metrics for NYC Cab dataset. Attention mechanism finds closer taxi zones to the stadiums compared to some node statistics such as betweenness centrality, clustering coefficient, and node degree. In other words, DyGED detects more crucial nodes for Baseball game detection.

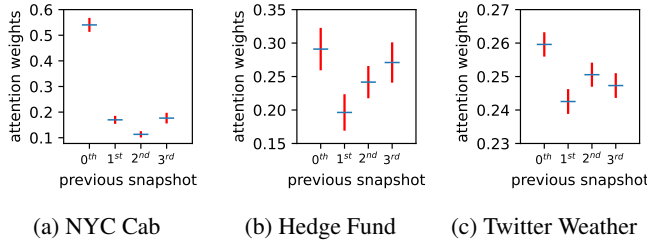

(a) NYC Cab   (b) Hedge Fund   (c) Twitter Weather

Figure 4: Illustration of the attention weights (normalized) of current and previous three snapshots for all datasets. While in NYC Cab, the current snapshot has more attention weights than the previous ones, attention weights in the Hedge Fund and Twitter datasets reveal more about the importance of temporal information. Results show that history plays a role in predicting the event. 0th refers to the current snapshot.

Figure 3 shows the top 10 most important 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, where nodes are words, our solution set contains "fire", "warm", "tree", and "snow" as the top words while the topology-based baselines have "weather", "update", "fire", and "barometer". The results show that 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. Similar to the node attention weights, the attention values here also signify the value of the information from the snapshots. We use three previous (i.e., $k = 4$) and the current snapshot in our experiments. Figure 4 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 several days, and thus weights are expected to be more uniformly distributed.

5.6 Event Embeddings and Diagnosis

Figure 5 shows graph embeddings produced by DyGED for our three datasets. Each point corresponds to a sequence of snapshots $G_{t-k:t}$. These are the same embeddings given as input to the MLP in our architecture. We set the number of dimensions h of the embeddings originally to 64 and then project them to 2-D using tSNE (Van der Maaten and Hinton 2008). We also annotate each embedding with the type-/cause of the event. For NYC Cab, we consider the stadium (*Yankees* or Mets) where the game takes place. For Hedge Fund, we distinguish between *up* and *down* shocks. For Twitter Weather, we use the classification from the US National Weather Service (*blizzard*, *hurricane*, *flood*, *wildfire*, *tornado*, *storm* and *extreme*).[7] Notice that this additional information is not used for event detection.

As desired, events often form clusters in the embedding space for all datasets. That illustrates the ability of DyGED to produce discriminative representations that enable the classifier to identify events. However, notice that events and non-events are not completely separated in 2-D embeddings produced by DyGED which outperform all the baselines. This suggests that the event detection problem on dynamic graphs might be non-trivial.

It is also interesting to analyze whether DyGED embeddings can be useful for *event diagnosis*, which consists of providing users with information that enables the identification of causes for the events (Julisch 2003; Jéron et al. 2006). More specifically, embeddings might capture not only whether an event occurs or not but also the nature of the event—i.e. events with the same cause are embedded near each other. Our results show that events of the same type/-cause tend to cluster in the embedding space. In particular, in the Twitter Weather dataset, events of type 'Extreme' and 'Storm' could be potentially diagnosed using a simpler nearest neighbors scheme.

---

[7]https://www.ncdc.noaa.gov/stormevents/pd01016005curr.pdf

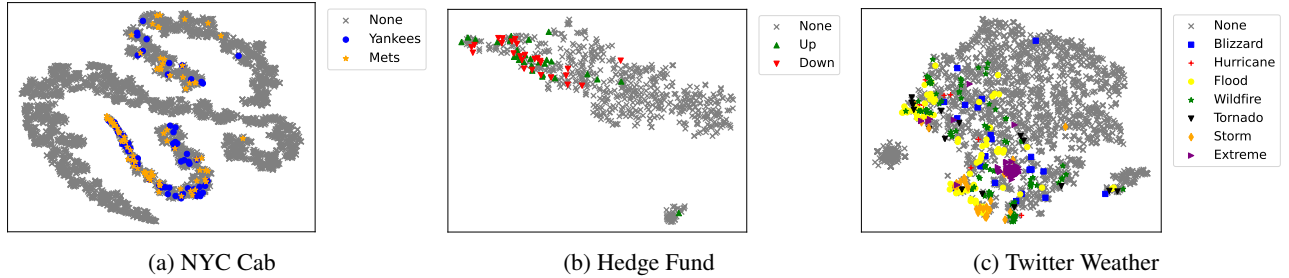| (a) NYC Cab | (b) Hedge Fund | (c) Twitter Weather |

Figure 5: An illustration of how DyGED graph embeddings support event diagnosis for three datasets. For each snapshot window $G_{t-k:t}$, we assign an event case (e.g. game at Yankees or Mets stadium for NYC Cab) or None. Results show that event causes tend to form clusters in the embedding for all datasets. Thus, events can be automatically diagnosed based on a few manually diagnosed ones in their embedding neighborhood. Better seen in color.

5.7 Event Detection Efficiency

Table 4 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).

	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 4: Testing times (in secs.) for all the methods with windows length *k+1* set to 4. DyGED is scalable and up to 206 times faster than the best baseline, ASTGCN.

5.8 Ablation Study

Feature Variation We also calculate three graph statistics as dynamic node features: node degree, betweenness centrality, and clustering coefficient. Table 5 shows results for static-only, dynamic-only, and static+dynamic node features for DyGED and the best baseline (ASTGCN) using all datasets. Results show that static features are often more relevant than dynamic ones—NYC Cab is the exception. Moreover, combining static and dynamic features often improves performance. We notice that for Twitter Weather (TW), static features (word embeddings) are quite expressive.

Pooling Operator Variation Mean and max operators are two of the most common pooling operators in convolutional networks (Goodfellow, Bengio, and Courville 2016). They have been used for down-sampling the data and their usefulness depends on the application and the dataset (Boureau, Ponce, and LeCun 2010). We compare these operators against self-attention graph pooling (Section 4.1). Table 6 shows that the self-attention operator (DyGED) outperforms the other operators for all datasets. These results highlight the importance of adaptive node weighting when capturing the macro dynamics for event detection, one of the essentials of our approach.

Method	NYC Cab	Hedge Fund	Twitter Weather
ASTGCN-S	0.903	0.753	0.774
DyGED-S	0.905	<u>0.787</u>	<u>0.810</u>
ASTGCN-D	0.894	0.741	0.747
DyGED-D	<u>0.911</u>	0.786	0.765
ASTGCN-SD	0.905	0.763	0.751
DyGED-SD	**0.925**	**0.798**	**0.815**

Table 5: AUC scores of DyGED and the best baseline method ASTGCN with different sets of node features. *-S*, *-D*, *-SD* suffixes are for static-only, dynamic-only, static+dynamic node features, respectively. TW is the Twitter Weather dataset.

Method	NYC Cab	Hedge Fund	Twitter Weather
DyGED-Mean	<u>0.879</u>	0.704	<u>0.773</u>
DyGED-Max	<u>0.880</u>	<u>0.729</u>	0.729
DyGED	**0.905**	**0.787**	**0.810**

Table 6: AUC scores of our event detection method DyGED equipped with different pooling operators. Attention pooling outperforms mean and max pooling for all datasets using our method DyGED. TW is the Twitter Weather data.

6 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 accuracy while being more scalable than the most effective one. We also showed how our method could be applied for event diagnosis as well as to provide interpretability via self-attention on nodes and snapshots. In future work, we want to develop hierarchical event detection architectures that are able to combine macro and micro dynamics. We are also interested in designing even more interpretable models for event detection on graphs so that the discovered events can be associated with subgraphs and their dynamics.

7 Acknowledgments

References

Aggarwal, C. C.; and Subbian, K. 2012. Event detection in social streams. In *SDM*.

Akoglu, L.; and Faloutsos, C. 2010. Event detection in time series of mobile communication graphs. In *Army science conference*.

Atefeh, F.; and Khreich, W. 2015. A survey of techniques for event detection in twitter. *Computational Intelligence*, 31(1): 132–164.

Bergmeir, C.; and Benítez, J. M. 2012. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191: 192–213.

Boureau, Y.-L.; Ponce, J.; and LeCun, Y. 2010. A theoretical analysis of feature pooling in visual recognition. In *ICML*, 111–118.

Chen, F.; and Neill, D. B. 2014. Non-parametric scan statistics for event detection and forecasting in heterogeneous social media graphs. In *SIGKDD*.

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

Duvenaud, D. K.; Maclaurein, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*, 2224–2232.

Errica, F.; Podda, M.; Bacciu, D.; and Micheli, A. 2020. A Fair Comparison of Graph Neural Networks for Graph Classification. In *ICLR*.

Fawcett, T. 2006. An introduction to ROC analysis. *Pattern recognition letters*, 27(8): 861–874.

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

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep learning*.

Gori, M.; Monfardini, G.; and Scarselli, F. 2005. A new model for learning in graph domains. In *IJNN*. IEEE.

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*.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

Ho, Y.; and Wookey, S. 2020. The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling. *IEEE Access*, 8: 4806–4813.

Hu, W.; Yang, Y.; Cheng, Z.; Yang, C.; and Ren, X. 2021. Time-Series Event Prediction with Evolutionary State Graph. In *WSDM*.

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

Jéron, T.; Marchand, H.; Pinchinat, S.; and Cordier, M.-O. 2006. Supervision patterns in discrete event systems diagnosis. In *WODES*.

Julisch, K. 2003. Clustering intrusion detection alarms to support root cause analysis. *TISSEC*.

Kashima, H.; Tsuda, K.; and Inokuchi, A. 2003. Marginalized kernels between labeled graphs. In *ICML*.

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

Kriege, N.; and Mutzel, P. 2012. Subgraph matching kernels for attributed graphs. In *ICML*.

Kriege, N. M.; Johansson, F. D.; and Morris, C. 2020. A survey on graph kernels. *Applied Network Science*, 5(1): 1–42.

Lee, J. B.; Rossi, R.; and Kong, X. 2018. Graph classification using structural attention. In *SIGKDD*.

Leetaru, K.; and Schrodt, P. A. 2013. Gdelt: Global data on events, location, and tone, 1979–2012. In *ISA annual convention*.

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*.

Lu, X.; and Brelsford, C. 2014. Network structure and community evolution on twitter: human behavior change in response to the 2011 Japanese earthquake and tsunami. *Scientific reports*, 4: 6773.

Lu, Y.; Wang, X.; Shi, C.; Yu, P. S.; and Ye, Y. 2019. Temporal network embedding with micro-and macro-dynamics. In *CIKM*, 469–478.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nguyen, G. H.; Lee, J. B.; Rossi, R. A.; Ahmed, N. K.; Koh, E.; and Kim, S. 2018. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*, 969–976.

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

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *ICML*.

Ning, Y.; Tao, R.; Reddy, C. K.; Rangwala, H.; Starz, J. C.; and Ramakrishnan, N. 2018. STAPLE: Spatio-Temporal Precursor Learning for Event Forecasting. In *SDM*.

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*.

Peel, L.; and Clauset, A. 2015. Detecting change points in the large-scale structure of evolving networks. In *AAAI*.

Pruthi, D.; Gupta, M.; Dhingra, B.; Neubig, G.; and Lipton, Z. C. 2019. Learning to deceive with attention-based explanations. *arXiv preprint arXiv:1909.07913*.

Ramakrishnan, N.; Butler, P.; Muthiah, S.; Self, N.; Khandpur, R.; Saraf, P.; Wang, W.; Cadena, J.; Vullikanti, A.; et al. 2014. 'Beating the news' with EMBERS: forecasting civil unrest using open source indicators. In *SIGKDD*.

Ranshous, S.; Shen, S.; Koutra, D.; Harenberg, S.; Faloutsos, C.; and Samatova, N. F. 2015. Anomaly detection in dynamic networks: a survey. *WIREs Computational Statistics*.

Rayana, S.; and Akoglu, L. 2015. Less is more: Building selective anomaly ensembles with application to event detection in temporal graphs. In *SDM*.

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

Rozenshtein, P.; Anagnostopoulos, A.; Gionis, A.; and Tatti, N. 2014. Event detection in activity networks. In *SIGKDD*.

Sakaki, T.; Okazaki, M.; and Matsuo, Y. 2010. Earthquake shakes Twitter users: real-time event detection by social sensors. 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.

Shervashidze, N.; Vishwanathan, S.; Petri, T.; Mehlhorn, K.; and Borgwardt, K. 2009. Efficient graphlet kernels for large graph comparison. In *AISTATS*.

Singer, U.; Guy, I.; and Radinsky, K. 2019. Node embedding over temporal graphs. *arXiv preprint arXiv:1903.08889*.

Skarding, J.; Gabrys, B.; and Musial, K. 2020. Foundations and modelling of dynamic networks using Dynamic Graph Neural Networks: A survey. *arXiv preprint arXiv:2005.07496*.

Staniford-Chen, S.; Cheung, S.; Crawford, R.; Dilger, M.; Frank, J.; Hoagland, J.; Levitt, K.; et al. 1996. GrIDS-a graph based intrusion detection system for large networks. In *NISSC*.

Taheri, A.; Gimpel, K.; and Berger-Wolf, T. 2019. Learning to Represent the Evolution of Dynamic Graphs with Recurrent Models. In *WebConf*.

Trivedi, R.; Farajtabar, M.; Biswal, P.; and Zha, H. 2019. DyRep: Learning Representations over Dynamic Graphs. In *ICLR*.

Van der Maaten, L.; and Hinton, G. 2008. Visualizing data using t-SNE. *JMLR*.

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

Wang, S.; Chen, Z.; Liu, B.; and Emery, S. 2016. Identifying search keywords for finding relevant social media posts. In *AAAI*.

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*.

Yan, S.; Xiong, Y.; and Lin, D. 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*.

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*.

Zhang, M.; Cui, Z.; Neumann, M.; and Chen, Y. 2018. An end-to-end deep learning architecture for graph classification. In *AAAI*.

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*.