# Scalable Dynamic Graph Representation Learning via Incremental Graph Embedding

Anonymous Author(s)

## ABSTRACT

Dynamic graph representation learning has broad applications, such as modeling the Internet of Things, social networks dynamics, and automated theorem proving. However, re-embedding all nodes at every time point in the graph sequence is computationally expensive. In practice, changes in graphs over time are small, affecting a relatively small subset of the nodes. Incrementally computing node embeddings significantly saves computation costs, which are particularly crucial in applications with long sequences and large graphs but small changes at each time point. We propose an algorithm to incrementally embed dynamic graphs based on graph convolutional neural networks. Every time step, we identify a small portion of the graph where the embedding should be updated while retaining previously computed embedding elsewhere. We propose an optimization objective to maintain the spatiotemporal consistency exhibited by dynamic graphs, which improves stability of the embedding. Our algorithm is applicable to any graph embedding method and can handle arbitrary graph changes, including adding and removing nodes. We apply our approach to benchmark graph prediction tasks, edge classification and link prediction. Our method demonstrates competitive predictive performance while updating only 10% or fewer node embeddings. Furthermore, our method uses 20× fewer parameters than state of the art baselines.

## CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Supervised learning*.

## KEYWORDS

Dynamic Graphs, Graph Embedding, Deep Learning, Graph Representation Learning, Graph Convolutional Networks

**ACM Reference Format:**

Anonymous Author(s). . Scalable Dynamic Graph Representation Learning via Incremental Graph Embedding. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages.

## 1 INTRODUCTION

Graph embedding methods aim to learn vector representations of graph data, which has attracted considerable attention in data science. Learned graph embeddings have shown to be highly valuable in various applications from edge classification in biological networks, to link prediction in social networks, to proof step classification in automated theorem proving. See comprehensive review papers on graph embedding [2, 9, 13] and the references therein for a complete review. The majority of the work on graph embedding have focused on static graphs, whereas many real world applications contain very rich temporal dynamics. Many approaches for dynamic embedding do not focus explicitly on scalability.

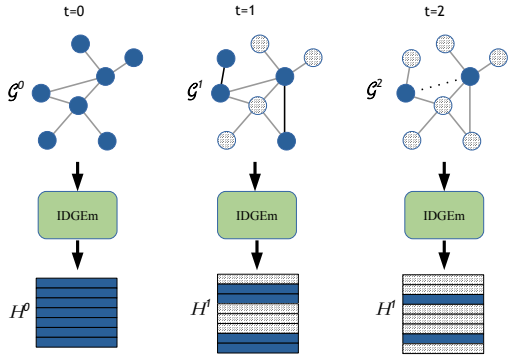


**Figure 1: As graph changes over time, only a subset of the embeddings are recomputed. Nodes that are endpoints of new or deleted edges are added to the node set $\mathcal{I}^t \subset V$ and the embedding of only these nodes are updated. Here, this set is shown in dark shading for each time point.**

The need to learn flexible representations of dynamic graphs gave rise to the flourishing research in *dynamic graph representation learning*. Specifically, the problem poses the question: given a sequence of graphs with changing nodes and edges, how to learn their graph representation? Several works have developed dynamic deep learning models for graph embedding, ranging from graph convolutional recurrent neural networks (RNNs) [3, 16, 21], to growing auto-encoders [7], to neural point processes [23, 26]. However, the main focus of these works is improving the accuracy of the learned embedding via better modeling of the graph evolution dynamics.

Apart from accuracy, *scalability* is a crucial technical challenge of dynamic graph representation learning. The computational complexity of graph embedding grows drastically with the length of the graph sequence. The number of node updates becomes computationally prohibitive for graphs with a large of number of nodes. In applications such as Internet of Things, the length of a graph sequence can be in the order of hundreds of thousands, and each graph can have millions of nodes. Therefore, obtaining high-quality graph representations while reducing the amount of computation is the key to deploying dynamic graph representation learning methods on real world industry-level platforms.

While there always exists a trade-off between accuracy and efficiency in machine learning models, it is particularly difficult to



*Conference'17, July 2017, Washington, DC, USA*



ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

strike a balance for graph sequences. First, dynamic graphs demonstrate strong temporal dependency. A small change in the embedding of the previous time step can have an immediate effect on the future embedding. Second, as the embedding encodes local topological information of the graphs, the updated embedding should also preserve such structural information while capturing the changes in nodes and edges. Lastly, the changes in both the nodes and the edges are highly irregular with adding and removing nodes, which is difficult to model using a single dynamical process.

In this paper, we aim to improve the scalability of dynamic graph representation learning methods via *incremental graph embedding*. We leverage the fact that the changes in a graph is local both spatially and temporally relative to the entire graph sequence. Therefore, we first identify a small portion of nodes that have experienced changes. We then update the embedding of the affected nodes while keeping the rest of the graph embedding unchanged, as shown in Figure 1. When applied to several large-scale dynamic graph data sets, we observe significant performance gain in computation with competitive prediction accuracy for benchmark graph embedding tasks. In summary, our contributions are:

- We propose an incremental graph embedding algorithm which first identifies a small portion of the graph as affected nodes and then updates the embedding for them accordingly.
- We conduct experiments on benchmark tasks using several large scale real world dynamic graph datasets and observe significant computational advantage with competitive predictive performance.
- We perform a careful ablation study to validate our approach.

## 2 RELATED WORK

Graph representation learning learns vector representation of graph data and has gained popularity in recent years, see several surveys on the topic [1, 25, 28]. *Dynamic* graph representation learning generalize graph embedding to temporal graphs. We refer readers to papers [2, 9, 13] and the references therein for a comprehensive review of the field. In general, dynamics graphs representation learning can be categorized into two types: *node dynamics* which characterize the evolution of node attributes; and *edge dynamics* which describe the changes in the graph structures.

For node dynamics, [16, 27] proposed to unify Graph Convolutional Networks (GCNs) and RNNs by replacing matrix multiplication in RNNs with graph convolution. [3, 21] proposed EvolveGCN, a two step approach to learn graph embedding at each time point and feed the embeddings into an RNN. [13] employed co-embedding methods to jointly model the sequence of interactions between users and items. For edge dynamics, DynGEM [7] propose to widen and deepen autoencoders for growing graphs. Others directly model the edge dynamics using RNNs [17], point processes [23], and triadic closure processes [29]. Recently, [6, 10] propose neural message passing algorithms to jointly capture node and edge dynamics. However, most existing methods emphasize the quality of the embedding for better accuracy rather than the efficiency. Most existing methods require re-embed the graph at every time point.

To scale up the models for learning graph representations, [4] proposed to optimizes the gradient computation in GCN in a single graph. But their method does not address the challenges in dynamic graphs, thus it can not reduce the number of node updates for embeddings across time points. Another line of research [5, 8, 19, 22] for structure-preserving dynamic graph embedding uses random walk based representations instead of deep neural networks. These methods only learn representations that encode graph structures rather than node attributes. For example, [5] usedd a skip-gram based graph representation and proposed to partially update the embedding for these nodes with the highest edge weights. Perhaps the work that is most related to our is [24], which derived an online learning algorithms to update the embedding from two attentive graph convolutional networks. However, their optimization objective is very specific for knowledge graphs and entity relationship embedding, which cannot be easily generalized for arbitrary graphs.

## 3 INCREMENTAL GRAPH EMBEDDING

### 3.1 Problem Formulation

We aim to learn informative representations for graph structured data in a dynamic setting; i.e., dynamic graphs. Formally, a dynamic graph is a sequence of graph snapshots $G^{1\to t} = [(X^1, A^1), \ldots, (X^t, A^t)]$ where $X \in \mathbb{R}^N$ is the node attributes and $A \in \mathbb{R}^{N\times N}$ is the adjacency matrix representing the graph structure.

Given a collection of such graphs, the problem of learning dynamic graph representation is thus defined as:

*Definition 3.1 (Dynamic Graph Representation Learning).* Given a dynamic graph $G^{1\to t}$, dynamic graph representation learning aims to learn a function that maps the graph sequence to a sequence of matrices,

$$f : G^{1\to t} \longrightarrow H^{1\to t}$$

where $H^t$ is all node embeddings of a graph at time $t$.

Note that an expressive function $f$ is supposed to capture not only the current graph topology and node attributes but also the spatial-temporal correlations between two consecutive graphs. The function $f$ is typically approximated by various deep neural networks, especially GCNs and RNNs. For example, EvolveGCN assumes the function $f$ to be decomposable over time $f = \prod_t f^t$, where each $f^t$ is a GCN and the learned embedding at different time steps are encoded with an RNN. These learned embeddings encode characteristic temporal and spatial properties of the original graph sequence and can be readily used for downstream graph prediction tasks such as edge classification or link prediction.

We focus on the *scalability* aspect of the representation learning problem rather than debating the type of architecture to use to approximate the embedding function. Existing methods for embedding dynamic graphs re-embed the entire graph at every time point, which suffers from high computational cost. However, in certain applications, graphs do not change drastically from one time point to the next. Accordingly, we propose an incremental approach which allows for more efficient computation by considering only the parts of graphs that have been changed. To this end, we propose to first identify the affected node set $\mathcal{I}^t$ that should be updated and then recompute the embeddings only for this subset, thus minimizing updates while obtaining the same quality of node embeddings.

Another important assumption we consider is that the embeddings of each node will not change dramatically from one time point to the next. Therefore, it is important to enforce the consistency of

node embeddings of each graph over different time steps. To achieve spatial-temporal smoothness of node embeddings, we present a graph regularization term jointing with a downstream task loss function to improve the stability of learned graph embeddings.

## 3.2 Identifying Affected Node Sets and Minimizing Embedding Updates

We learn nodes representations using a multi-layer GCN [10] comprised of graph convolution operations at each layer. First, we outline the general case for GCN without incremental updates. At the input layer $l = 0$, the node representation $H_0^t$ is given by the node features $X^t$. In subsequent layers, the representation is updated by layer weights $W_l^t$ and activation $\sigma(\cdot)$. That is, the embedding matrix is updated using the following update rule:

$$H_l^t = \text{GCN}\left(A^t, H_{l-1}^t, W_l^t\right) = \sigma\left(\hat{A}^t H_{l-1}^t W_l^t\right). \tag{1}$$

For simplicity, we denote the embedding at the last layer simply as $H^t$ and specify the update across all $L$ layers as

$$H^t = \text{GCN}_L\left(A^t, X^t, W^t\right) \tag{2}$$

In this formulation, the representation $h_u^t$ for all $N^t$ nodes are updated at every time point and every layer. We posit such an exhaustive update is not necessary for every node at each time. Instead, for a dynamic graph $\mathcal{G}^{0,\ldots,T}$, a subset of nodes $\mathcal{I}^t$ to be updated can be identified based on changes in the graph. We define $\Delta\mathcal{G}^t$ generally as graph updates from $t-1$ to $t$. These updates are comprised of changes in the structure $|A^t - A^{t-1}|$ and node features $|X^t - X^{t-1}|$. The set $\mathcal{I}^t$ contains the nodes that are endpoints of new or deleted edges and nodes $v$ where $|x_v^t - x_v^t| > \tau$ for a threshold $\tau$.

Once the affected node set $\mathcal{I}^t$ is determined, the embedding only for nodes in that set are updated. Therefore, the IDGEm update rule is given by the layer-wise update rule

$$H_{[\mathcal{I}]}^t = \text{GCN}_L\left(A_{[\mathcal{I}]}^t, X_{[\mathcal{I}]}^t, W^t\right). \tag{3}$$

Here, $A_{[\mathcal{I}]}^t$ and $X_{[\mathcal{I}]}^t$ indicate we index each matrix by the nodes in $\mathcal{I}^t$. We have dropped the time index for $\mathcal{I}^t$ for simplicity of notation, but the set will change for each $t$ as determined by $\Delta\mathcal{G}^t$. For nodes not included in the affected node set, the embedding is carried over from the previous time point without update.

Because the model weights depend only on the embedding dimension and the hidden size, these update rules allow for the size of the graph to change over time. A basic GCN model that learns node representations at every time point can also handle these changes. However, in many cases an incremental approach like IDGEm can provide very efficient computation while still learning accurate node representations. For dynamic graphs that change over time by adding very few nodes at time to very large graphs, IDGEm is especially advantageous. For example, social networks may be very large and change slowly over time. New embeddings would be learned only for new nodes and the nodes to which they are connected, avoiding recomputing the remaining nodes in the graph. For shrinking graphs, the embedding for any removed nodes could be removed from the embedding matrix $H^t$ as long as the indexing for remaining nodes is taken into account. The embedding of nodes previously connected to removed nodes would be updated.

## 3.3 Learning

We formulate the problem of node representation learning using the IDGEm model as a supervised learning problem on a downstream task on the dynamic graph. A simple MLP classifier uses the learned representation for predictions on the downstream task. In the case of link prediction, the classifier takes the embedding of pairs of nodes and predicts if a link is present. For edge classification, the classifier predicts the class of an existing edge.

To train the model, we propose a multi-term loss function that uses a supervised signal on the downstream task as well as spatial and temporal smoothing terms. Specifically, we prose to minimize

$$\mathcal{L}(H^t) = \sum_{v \in V} \sum_{u \in \mathcal{N}(v)} \mathcal{L}_{\text{spatial}}(h_u^t, h_v^t) + \mathcal{L}_{\text{temporal}}(h_v^{t-1}, h_v^t) + \mathcal{L}_{\text{task}}(h_v^t) \tag{4}$$

The *spatial smoothing* term $\mathcal{L}_{\text{spatial}}$ encourages learning node representations that are similar for connected nodes. This term assumes that similar nodes are connected, a reasonable assumption for many applications. The *temporal smoothing* term $\mathcal{L}_{\text{temporal}}$ promotes learning representations for each node that are similar to the representation learned in previous time steps. In particular, we use the following loss function

$$\mathcal{L}(H^t) = \sum_{v \in V} \sum_{u \in \mathcal{N}(v)} |h_u^t - h_v^t|^2 + \frac{|h_v^{t-1} - h_v^t|}{\|h_v^{t-1}\|} + \mathcal{L}_{\text{task}}(h_v^t) \tag{5}$$

The task-specific loss term $\mathcal{L}_{\text{task}}(h_v^t)$ will depend on the downstream task used for training. Since we only update the embedding for nodes in $\mathcal{I}^t$, the outer sum in (4) need only be computed over this subset of nodes. Thus, the loss can be minimized taking into consideration only the nodes in $\mathcal{I}^t$ and their first neighbors

$$\mathcal{L}(H^t) = \sum_{v \in \mathcal{I}^t} \mathcal{L}_{\text{task}}(h_v^t) + \mathcal{L}_{\text{temporal}}(h_v^{t-1}, h_v^t) + \sum_{u \in \mathcal{N}(v)} \mathcal{L}_{\text{spatial}}(h_u^t, h_v^t) \tag{6}$$

# 4 EXPERIMENTS

We evaluate our model performance on benchmark tasks of edge classification and link prediction on several dynamic graph datasets. We also investigate its robustness through an ablation study on our loss and present a qualitative analysis of the embedding space.

## 4.1 Baselines

We compare our incremental embedding method to a set of baselines, (1) StaticGCN – a naive baseline using a vanilla GCN that learns an embedding for each node at every time point. The embedding update is given by Equation (1); (2) TemporalGCN – a more sophisticated baseline where node representations across time are not independent. Node representations are first updated with a multi-layer GCN with output $H_L^t$. Then, these node representations are used as the inputs to a recurrent layer with LSTM units. This model captures the dynamics of the embedding evolution but also recomputes the embedding for every node in the graph; and (3) EvolveGCN [21] which learns node embeddings for dynamic graphs by updating the GCN parameters over time according to dynamics learned by an RNN. We consider both variants, EvolveGCN-h and EvolveGCN-o and use the hyperparameters provided by the configuration files in the publicly available code.

**Table 1: Model parameters for each method in thousands**

| EvolveGCN-h | EvolveGCN-o | StaticGCN | TemporalGCN | IDGEm |
|---|---|---|---|---|
| 3,631 | 2,958 | 127 | 140 | 127 |

## 4.2 Datasets

We evaluate on several dynamic graph datasets from the SNAP [15] dataset collection. We follow the train/validation/test splits proposed by [21]. We train the models on the first several time steps, then validate and test on subsequent time steps.

The Bitcoint-OTC **BT-OTC** dataset [11, 12] provides transactions between users of the Bitcoin-OTC platform. The graphs have directed, weighted edges. The edge weights indicate a trust rating from one user to another; negative weights indicate distrust while positive ones indicate trust. For edge classification, we predict two classes, untrustworthy (class 0) and trustworthy (class 1).

The Autonomous Systems **AutSyst** dataset [14] provides Internet router subgraph configurations over 785 days. We aggregate days together to produce 100 time steps and use this dataset for link prediction. The links are unweighted and undirected.

The UC-Irvine Message **UCI** dataset [20] encodes messages sent between users of a student social network. Edges are directed from the sender to the receiver.

We run each model setting with three model parameter random initialization seeds. We evaluate the model by observing the graph for a time window $t = [t_0, t_W]$, updating the embeddings according to model update rules, and performing a predictive task on the graph at $t = t_W + 1$. We train all models with sliding window across the training portion and report results averaged over all windows on the test portion. The embeddings are then used for downstream prediction tasks on the nodes, using the learned vector representations as inputs to a simple two-layer MLP classifier trained jointly with the embedding model.

## 4.3 Prediction Tasks

The goal of the *edge classification* task is to assign a label to an edge that appears at time $t$. The endpoints of the edge are known but the label is not. For example, in the BT-OTC dataset, a transaction between two uses occurs at time $t$ and the goal is to classify this transaction as trustworthy or untrustworthy. For this task, the embedding of the source node and the embedding of target node are concatenated to serve as the input to the classifier. We use cross entropy classification loss for the task-specific loss, $\mathcal{L}_{\text{task}} = -\log\left(\frac{\exp(x^t_{uv}[c])}{\sum_i \exp(x^t_{uv}[c])}\right)$, with the classifier output for edge $(u, v)$ represented as $x^t_{uv}$ and indexed by the true class $c$.

In *link prediction*, the goal is to predict the presence or absence of an edge between any two nodes in the graph at time $t$. For this task, the embeddings of every pairwise combination of nodes are concatenated and used as the input to a classifier that assigns a probability to the presence of an edge between the two nodes. We use binary cross entropy as the task loss term.

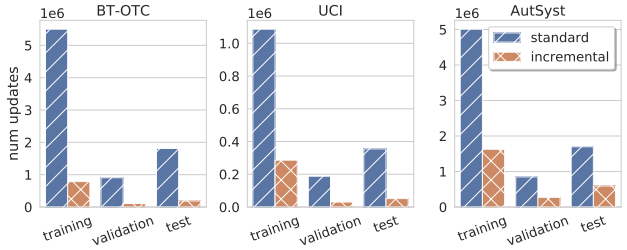


**Figure 2: Updates for our incremental method and *standard* methods that update all node embeddings at every time point. IDGEm computes only 10-35% of the total nodes.**

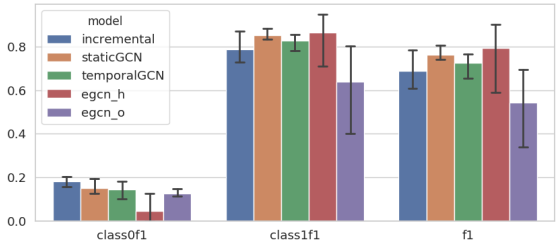


**Figure 3: BT-OTC edge classification results. F1 classification score for the minority class (0) and majority class (1) and the average across all classes is reported.**

## 4.4 Accuracy and Scalability

We first investigate the number of node embedding updated for IDGEm compared to baseline methods, which we designate as *standard* since they recompute the embedding for all nodes. We illustrate the savings in node embedding updates across datasets in Figure 2. The fraction of node embedding updates for incremental methods compared to standard methods are between about 10% and 35%, depending on the dataset.

For edge classification, we evaluate each model using the F1 score for classifying each class and the average performance across classes, shown in Figure 3. The results show EvolveGCN modes have larger variance between random model parameter initialization while IDGEm and the baseline models StaticGCN and TemporalGCN trained with the incremental loss have much small variance. While the performance of the models are comparable, IDGEm updates only 10% of node embeddings for the test sequence compared to all other models Likewise, number of parameters is an order of magnitude smaller for IDGEm and the corresponding baseline models compared to EvolveGCN models, as shown in Table 1.

For link prediction, we report two measures, Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR) as in [21]. MAP incorporates recall and precision at various prediction thresholds, capturing the precision-recall curve for the binary prediction task. MRR captures the ranking of ground-truth links for each node among possible links. For both metrics, higher values correspond to better predictions. We show the results of these experiments in Table 2. We see a similar pattern of high variance in the EvolveGCN methods while performance remains comparable across methods.

**Table 2: Link prediction results in terms of Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR); higher values correspond to better predictions.**

| | BT-OTC | | UCI | | AutSyst | |
|---|---|---|---|---|---|---|
| | MAP | MRR | MAP | MRR | MAP | MRR |
| EvolveGCN-h | 1.796e−3 ± 7.85e−4 | 0.051 76 ± 0.0272 | 9.135e−3 ± 1.57e−3 | 0.1102 ± 6.36e−3 | 9.768e−2 ± 5.21e−2 | 0.260 945 ± 8.77e−2 |
| EvolveGCN-o | 1.904e−3 ± 1.50e−3 | 0.086 90 ± 3.07e−3 | 1.008e−2 ± 5.43e−3 | 0.1042 ± 1.81e−3 | 6.293e−2 ± 3.31e−2 | 0.246 790 ± 6.42e−2 |
| Static GCN | 8.273e−4 ± 1.517e−4 | 0.088 17 ± 2.01e−3 | 8.353e−3 ± 8.66e−4 | 0.1117 ± 1.77e−3 | 1.878e−2 ± 1.13e−2 | 0.174 424 ± 8.08e−3 |
| Temporal GCN | 9.352e−7 ± 2.157e−7 | 4.47e−4 ± 1.16e−4 | 2.711e−3 ± 1.53e−4 | 0.092 86 ± 1.13e−2 | 1.848e−2 ± 7.24e−3 | 0.143 708 ± 4.51e−2 |
| IDGEm | 5.257e−4 ± 2.501e−4 | 0.086 89 ± 2.49e−3 | 8.018e−3 ± 4.81e−4 | 0.1077 ± 2.47e−3 | 1.626e−2 ± 2.09e−3 | 0.158 667 ± 1.75e−2 |

Again, the number of parameters is much smaller for IDGEm and the number of node embedding updates is 10-35% of the total.

## 4.5 Incremental Loss Ablation Study

We propose the supervised incremental loss function in (4) which regularizes the task-specific performance with spatial and temporal smoothing. The spatial smoothing term promotes learning similar node representations for connected nodes. The temporal smoothing term keeps node representations similar at consecutive time points.

We conduct an ablation study on the effect of these regularization terms on the performance of edge classification (see Figure 4). To validate the utility of the smoothing terms, we train the IDGEm model while removing each of the smoothing terms individually and with no smoothing. For each setting, we train the model three times, each time with a different random initialization of parameters. Though the *average* performance for all settings is similar, the model trained without any smoothing and trained without spatial smoothing exhibits higher variance than when trained with the full loss or with spatial smoothing only. Similar results are seen in the performance for the majority class (class 1). However, the latter two loss settings perform better in the classification of the minority class (class 0) while still reducing the variance. This may indicate that spatial smoothing has a greater regularization effect and avoids the model overfitting to the majority class.

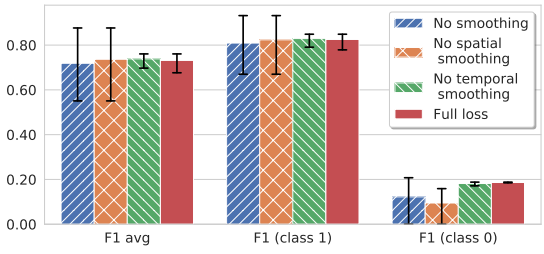


**Figure 4: Edge classification (BT-OTC) performance of IDGEm for different loss formulations. No smoothing is the task-specific supervised loss. No spatial- and No temporal- removes one of the terms in the loss function (4).**

## 4.6 Analysis of Embedding Space

To gain a better understanding of the embeddings learned by each model, we use tSNE [18] to visualize the embedding space. For edge classification in the BT-OTC dataset, we visualize the input to the classifier. Namely, for each edge to be classified, the representation of the source and target nodes are concatenated together. The resulting tSNE plot is shown in Figure 5 with the majority class shown in gray and the minority class (untrustworthy) in red. We show the space for the first five time steps of the test set for each model. These visualizations suggest agreement with the quantitative results shown in our experiments. The learned representations in StaticGCN and IDGEm also look similar to each other, supporting the assumption that updating only the embeddings for affected nodes is sufficient for learning a reasonable node representation.

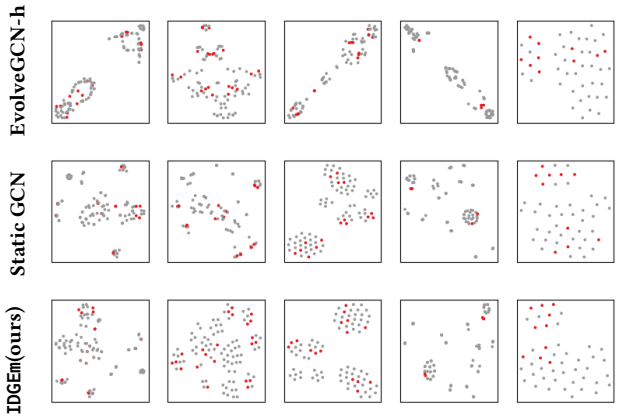


**Figure 5: tSNEs of embeddings for 5 time points. Each point is embedding for node pairs that are endpoints to edges in edge classification task for Bitcoin-OTC dataset. Ground-truth classes are differentiated by color.**

## 5 CONCLUSIONS

In this work, we present a novel approach for computing scalable dynamic graph embeddings by minimizing the number of updates while maintaining performance. To this end, we first identify the set of affected nodes and update embedding only for these affected nodes. We also introduce spatial-temporal regularization to enforce consistency of learned node embeddings, which help significantly stabilize the learned graph embeddings over time. Our experimental results on benchmark datasets demonstrate that our proposed model yields comparable performance to traditional methods while only exhibiting ≤10% of their computational costs for re-computing node embeddings.

# REFERENCES


[1] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

[2] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[3] Jinyin Chen, Xuanheng Xu, Yangyang Wu, and Haibin Zheng. 2018. GC-LSTM: Graph Convolution Embedded LSTM for Dynamic Link Prediction. *arXiv:1812.04206* (2018).

[4] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *Proc. of KDD*. 257–266.

[5] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. 2018. Dynamic Network Embedding : An Extended Approach for Skip-gram based Network Embedding. In *Proc. of IJCAI*. 2086–2092.

[6] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *Proc. of ICML*. 1263–1272.

[7] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. Dyngem: Deep embedding method for dynamic graphs. *arXiv:1805.11273* (2018).

[8] Tomasz Kajdanowicz, Kamil Tagowski, Maciej Falkiewicz, Piotr Bielak, Przemysław Kazienko, and Nitesh V. Chawla. 2019. Incremental embedding for temporal networks. *arXiv:1904.03423* (2019).

[9] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2019. Relational representation learning for dynamic (knowledge) graphs: A survey. *arXiv:1905.11485* (2019).

[10] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. of ICLR*.

[11] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. 2018. Rev2: Fraudulent user prediction in rating platforms. In *Proc. of ACM International WSDM Conference*. 333–341.

[12] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In *Proc. of ICDM*. 221–230.

[13] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proc. of KDD*. 1269–1278.

[14] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. of KDD*. 177–187.

[15] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[16] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *Proc. of ICLR*.

[17] Yao Ma, Ziyi Guo, Zhaochun Ren, Eric Zhao, Jiliang Tang, and Dawei Yin. 2018. Streaming Graph Neural Networks. *arXiv:1810.10627* (2018). arXiv:1810.10627.

[18] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[19] Sedigheh Mahdavi, Shima Khoshraftar, and Aijun An. 2018. dynnode2vec: Scalable dynamic network embedding. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 3762–3765.

[20] Pietro Panzarasa, Tore Opsahl, and Kathleen M Carley. 2009. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology* 60, 5 (2009), 911–932.

[21] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, and Charles E Leisersen. 2019. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. *arXiv:1902.10191* (2019).

[22] Hooman Peiro Sajjad, Andrew Docherty, and Yuriy Tyshetskiy. 2019. Efficient Representation Learning Using Random Walks for Dynamic Graphs. *arXiv:1901.01346* (2019).

[23] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *Proc. of ICLR*.

[24] Tianxing Wu, Arijit Khan, Huan Gao, and Cheng Li. 2019. Efficiently Embedding Dynamic Knowledge Graphs. *arXiv:1910.06708* (2019).

[25] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv:1901.00596* (2019).

[26] Shuai Xiao, Junchi Yan, Xiaokang Yang, Hongyuan Zha, and Stephen M Chu. 2017. Modeling the intensity function of point process via recurrent neural networks. In *Proc. of AAAI*.

[27] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv:1709.04875* (2017).

[28] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2018. Deep Learning on Graphs: A Survey. *arXiv:1812.04202* (2018).

[29] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *Proc. of AAAI*.