作者提出了一种在时间图中嵌入节点的方法。该算法可学习时间图的节点和边沿随时间的演变，并将此动态结合到用于不同图预测任务的时间节点嵌入框架中

本文提出了一种联合损失函数，该联合损失函数通过学习组合节点的历史时态嵌入来创建节点的时态嵌入，以便针对每个给定任务进行优化（例如，链接预测）。使用静态节点嵌入对算法进行初始化，然后将其在不同时间点的节点表示上对齐，并最终在联合优化中适合给定任务。

Node Embedding over Temporal Graphs

Uriel Singer
Technion, Israel Institute of Technology
Haifa, Israel
urielsinger@campus.technion.ac.il

Ido Guy
eBay Research
Netanya, Israel
idoguy@acm.org

Kira Radinsky
Technion, Israel Institute of Technology
Haifa, Israel
kirar@cs.technion.ac.il

这篇文章借鉴NLP中机器翻译对齐的思想，代码其实就一行可以搞定

ABSTRACT

提出可以学习节点和边演化的算法
并在面对不同预测任务时将之合并入节点嵌入框架

In this work, we present a method for node embedding in temporal graphs. We propose an algorithm that learns the evolution of a temporal graph's nodes and edges over time and incorporates this dynamics in a temporal node embedding framework for different graph prediction tasks. We present a joint loss function that creates a temporal embedding of a node by learning to combine its historical temporal embeddings, such that it optimizes per given task (e.g., link prediction). The algorithm is initialized using static node embeddings, which are then aligned over the representations of a node at different time points, and eventually adapted for the given task in a joint optimization. We evaluate the effectiveness of our approach over a variety of temporal graphs for the two fundamental tasks of temporal link prediction and multi-label node classification, comparing to competitive baselines and algorithmic alternatives. Our algorithm shows performance improvements across many of the datasets and baselines and is found particularly effective for graphs that are less cohesive, with a lower clustering coefficient.

ACM Reference Format:
Uriel Singer, Ido Guy, and Kira Radinsky. 2019. Node Embedding over Temporal Graphs. In *Proceedings of* . ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Web is a growing universe of interlinked pages, social ties, and a fund of human knowledge accumulated over the years. The understanding of Web change has implications for tools that are designed to help people interact with dynamic content, such as search engines and recommender systems. Understanding the development of large graphs over time bears significant importance to understanding community evolution and identifying deviant behavior. For example, identifying nodes that have an unusual structure change over time might indicate an anomaly or fraud [1]. Another structure change can help identify new communities in a social network and thus can be used to improve social recommendations of friends [36, 40] or detect roles in a network [41]. Other applications can optimize the network's growth by building smarter caching mechanisms.

Many important tasks in static network analysis focus on predictions over nodes and edges. Node classification assigns probabilities to a set of possible labels. For example, a person's role in a social network. Another important task is link prediction [30], which involves assigning a probability to an edge. For example, predicting whether two users in a social network are friends. In this paper, we focus on dynamic predictions of future interactions that involve structural changes. For example, predicting a person's future role or whether two users will become friends next year. （P4公式2）

Classic techniques for node and link prediction aim to define a set of features to represent the vertices or edges. These are subsequently used in a supervised learning setting to predict the class. Commonly, to learn the features, linear and non-linear dimensionality reduction techniques such as Principal Component Analysis (PCA) are applied [4, 42, 46]. They transform the graph's adjacency matrix to maximize the variance in the data. More recent approaches, which have shown substantial performance improvement, aim at optimizing an objective that preserves local neighborhoods of nodes [39, 45]. Lately, embedding-based approaches [14, 49] showed state-of-the-art performance for graph prediction tasks. For example, node2vec [14] optimizes for embedding, where nodes that share the same network community and/or similar roles have similar representations in a latent space of a lower dimension. It performs biased random walks to generate neighbors of nodes, similarly to previous work in natural language processing [33]. The use of pre-computed embeddings as features for supervised learning algorithms allows to generalize across a wide variety of domains and prediction tasks.

In this work, we extend the prior embedding-based approaches, to include the network's temporal behavior. Intuitively, if node2vec mimics word embeddings as a representation for node embedding, we present an algorithm that mimics sentence embedding [35] as an extended representation of a node. Each word in the sentence is a temporal representation of the node over time, capturing the dynamics of its role and connectivity. We propose tNodeEmbed, a semi-supervised algorithm that learns feature representations for temporal networks. We optimize for two objective functions: (1) preserving static network neighborhoods of nodes in a d-dimensional feature space and (2) preserving network dynamics. We present an optimization function to jointly learn both (1) and (2). We achieve (1) by leveraging static graph embedding. Specifically, in this work we perform experiments with node2vec embeddings, which have the advantage of being unsupervised and scaling well over large graphs [14]. As graph embeddings do not preserve coherence, i.e., each training of node embedding by (1) on the same graph can provide different node embeddings, we explore several approaches for aligning the graph representation to only capture true network dynamics rather than stochasticity stemming from the embedding training. We then achieve (2) by creating a final embedding of a

, ,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

IJCAI

node by a jointly learning how to combine a node's historical temporal embeddings, such that it optimizes for a given task (e.g., link prediction).

In our experiments, we present results for two types of predictions: (1) temporal link prediction, i.e., given a pair of disconnected nodes at time t, predict the existence of an edge between them at time $t+n$; (2) multi-label node classification, i.e., given a node, predict its class. We experiment with a variety of real-world networks across diverse domains, including social, biological, and scholar networks. We compare our approach with state-of-the-art baselines and demonstrate its superiority by a statistically significant margin. In addition, we observe that our algorithm reaches better prediction performance on graphs with a lower clustering coefficient. We hypothesize that when the network is more cohesive, the contribution of the network dynamics to the prediction is lower. Intuitively, as graph generation follows the preferential attachment model [3], new nodes will tend to attach to existing communities rendering them more cohesive. The usage of dynamics is especially important in graphs that have not yet developed large cohesive clusters. We show analysis on several synthetic graphs mimicking different growth dynamics to support this hypothesis. We also perform a broad analysis of the algorithm parameters and observe that our alignment approach significantly improves the performance.

The contribution of this work is threefold:

- We propose tNodeEmbed, an algorithm for feature learning in temporal networks that both optimizes for preserving network structure and network dynamics. We present results for both node classification and edge prediction tasks.
- We study when network dynamics brings most value for network structure prediction. We identify that in order to learn it successfully, a minimum historical behavior of a node is needed. Additionally, we observe that predictions over graphs of higher clustering coefficients are not significantly improved by incorporating network dynamics into node embeddings.
- We empirically evaluate tNodeEmbed for both edge and node prediction on several real-world datasets and show superior performance.

2 RELATED WORK

Various works have explored the temporal phenomena in graphs: Leskovec et al. [25] empirically studied multiple graph evolution process over time via the average node degree. Others studied network evolution [24], knowledge graph dynamics [47], and information cascades on Facebook and Twitter [6, 23].

Temporal graph behavior has been studied in several directions: some works [22, 27, 29, 58] focused on temporal prediction problems where the input is a graph. They studied numerous deep-learning approaches for representing an entire graph and minimizing a loss function for a specific prediction task. Other methods [28, 38, 55] directly minimized the loss function for a downstream prediction task without learning a feature-representation. Most of these methods do not scale well to larger graphs due to high training time requirements, as they build models per node [28, 55] or models with parameters that depend on the amount of nodes [48]. Other models do not scale well across multiple time-stamps [28] or scale well but at the cost of a relatively low performance [9].

To overcome the scaling drawbacks, one of the common approaches today for node and edge embedding focuses on feature learning for graph prediction tasks. Feature learning in static graphs is typically done by using the graph matrix representation and preforming a dimensionality reduction, such as spectral clustering or PCA [4, 42, 46, 53]. Yet, eigendecomposition of the graph matrix is computationally expensive, and therefore hard to scale for large networks.

Most recent approaches [14, 39] for feature learning in graphs aim to find a representation for nodes by learning their embeddings via neural networks, with no need for manual feature extraction. These methods have shown state-of-the-art results for both node classification and edge prediction. However, such static network models seek to learn properties of individual snapshots of a graph. In this work, we extend the state-of-the-art feature-learning approach to include the temporal aspects of graphs. Some recent works have attempted to improve the static embeddings by considering historical embedding of the nodes and producing more stable static embeddings [13]. In this work, we aim at leveraging the node and edge dynamics to yield better and more informative embeddings, which can later be used for temporal prediction tasks. Intuitively, rather than smoothing out the dynamics, we claim it brings high value for the embeddings and predictions.

Temporal predictions have been explored in the past. Several studies applied graph convolution networks for static graphs with changing node attributes, for example, for learning human movement [11, 20, 52], traffic forecasting [8, 54] and other urban dynamics [50]. In these problems, the graph (e.g., the human skeleton or road map) is static and the goal is to predict a quality related to a node, such as how many cars will traverse a certain junction. More closely-related methods to our problem domain aimed to learn single-time representations and then statically combine them focusing on a specific task. For example, Dunlavy et al. [10] explored forecasting which edges will be formed at a future time point. Their method first collapsed multi-year data into a single matrix using weights that apply a decay factor to earlier time points. Matrix factorization methods were then applied to learn a representation of the aggregated matrix that is optimized for the temporal link prediction task. Similarly, Yu et al. [55] learned how to combine representations of matrices that represent temporal snapshots by imposing a polynomial relation among them. In this work, we show that learning how to combine the historical dynamics, rather than merely applying a weighting scheme, brings significant empirical improvements.

过去：静图时域预测

Several other approaches have been recently suggested for temporal graph representations. Nguyen et al. [34] proposed a method for continuous dynamic embedding, based on random walks with 'chronological' paths that can only move forward in time. Their approach was shown to outperform static baselines, including DeepWalk [39] and node2vec [14]. Zuo et al. [59] also proposed a continuous method, with neighborhoods generated by modeling Hawkes processes [16]. The dynamic triad algorithm [56] considered the addition of a third edge among three nodes in the graph (dynamic triad closure) to capture network dynamics over time and learn from one time point to the next. Other temporal representations include the application of generalized singular value decomposition to consecutive time steps [57] and the use of node attribute

similarity to optimize representation over time [26]. Recent surveys provide good summaries and additional details [7, 12, 15]. Our approach is unique in providing an end-to-end architecture that can be jointly optimized to a given task. In addition, our evaluation shows the superiority of our approach for the relevant baselines over a variety of datasets for both the temporal link prediction and node classification tasks.

3 FEATURE LEARNING FRAMEWORK

Let $G = (V, E)$ be a graph where each temporal edge $(u, v)_t \in E$ is directed from a node u to a node v at time t. We define a temporal graph, $G_t = (V_t, E_t)$, as the graph of all edges occurring up to time t. We define the evolution of G_t over time by the set of graph snapshots, $G_{t_1}, ..., G_{t_T}$, at T different time steps, $t_1 < ... < t_T$. In this work, we propose an algorithm that learns the evolution of a temporal graph's nodes and edges over time in an end-to-end architecture for different prediction tasks.

Our goal is to find for each node $v \in V$ at time T a feature vector $f_T(v)$ that minimizes the loss of any given prediction task. We consider two major prediction tasks: node classification and link prediction. For the node classification task, we consider a categorical cross-entropy loss, i.e.:

$$L_{task} = -\sum_{v \in V} \log Pr(class(v)|f_T(v))$$

where $class(v)$ is the class label of a node v. For the link prediction task, we consider a binary classification loss, i.e.:

$$L_{task} = -\sum_{v_1, v_2 \in V} \log Pr((v_1, v_2) \in E_t|g(f_T(v_1), f_T(v_2)))$$

where g can be any function. In this work, we consider the concatenation function. It is important to note that, without loss of generality, other tasks with corresponding loss functions can be supported in our framework.

We wish to leverage the set of graph snapshots, $G_1, \ldots, G_T$ to learn a function F_T, s.t.: $f_T(v) = F_T(v, G_1, \ldots, G_T)$, that best optimizes for L_{task}. To learn node dynamics, we formalize the embedding of a node v at time $t+1$ in a recursive representation:

$$f_{t+1}(v) = \sigma(Af_t(v) + BQ_tR_tv) \quad (1)$$

where $f_0(v) = \vec{0}$, A, B, R_t and Q_t are matrices that are learned during training, v is a one-hot vector representing a node, and σ is an activation function. We consider several such functions and further discuss them in Section 3.3.

We formulate the final temporal embedding learning problem as an optimization problem minimizing the following loss function:

$$L = \min_{A, B, Q_1, \ldots Q_T, R_2, .. R_T} L_{task} \quad (2)$$

We optimize this equation using Adam [21] over the model parameters defining the embedding f_T, i.e., $A, B, Q_1, \ldots, Q_T, R_2, ..R_T$.

Intuitively, one can interpret Q_t as a matrix of static node representation in a specific time t. Minimizing the loss for Q_t can be thought of as optimizing for a node embedding of a static graph snapshot at a previous time point, such that the final f_T optimizes L_{task}. We consider several initialization mechanisms for Q_t. Specifically, we learn the representation by sampling neighboring nodes from the corresponding graph G_t at time t, for all nodes, while minimizing the following loss function:

$$-\sum_t \sum_{v_t \in V_t} \log Pr(N(v_t)|Q_t v_t) \quad (3)$$

where $N(v)$ is a network of neighbors of a node v generated through some network sampling strategy. In Section 3.1, we discuss in detail the initialization procedure.

We also require that an alignment between consecutive time steps of the static embeddings is preserved by enforcing R_t as a rotation matrix. R_t therefore minimizes:

$$\sum_t \|R_{t+1}Q_{t+1} - Q_t\| + \lambda\|R_{t+1}^T R_{t+1} - I\| \quad (4)$$

where R_1=I and λ is a hyper-parameter of the model (in our experiments, we set λ=1). The first element ensures the embedded representations of a node between two consecutive time steps are similar, while the second element forces R to be a rotation matrix. Section 3.2 provides additional details about the alignment process.

This end-to-end procedure enables us to learn how to combine a node's historical temporal embeddings into a final embedding, such that it can optimize for a given task (e.g., link prediction), as defined by L_{task}. The rest of the section is structured as follows: we first discuss the initialization procedure for Q_t (Section 3.1). We then discuss in detail the optimization of R_t (Section 3.2). We finish the discussion of the framework by discussing the optimization using a deep learning architecture (Section 3.3), which creates a final embedding of a node by learning how to combine its historical temporal embeddings, such that it optimizes per given task.

3.1 Initialization using Static Node Embeddings

The matrix Q_t plays a key role in constructing the final embedding of a node based on its historical snapshots (see eq. 1), which is then optimized for the end-goal task (see eq. 2). In this section, we discuss the initialization procedure for Q_t.

Several prior works [14, 39, 45] studied node embedding and reported good performance on *static* graphs, i.e., graphs that either represent a specific point in time (snapshot) or do not change over time. The architecture presented in this paper can use any of the known approaches for node embedding over static graphs. Specifically, we opted to work with node2vec [14], which achieved state-of-the-art performance on various benchmarks.

Node2vec is based on word2vec [33], which is a framework for feature learning representation of words. The framework receives as input a text corpus and outputs an embedding in a low-dimensional space of size d (a hyper-parameter) for each word. Node2vec generalizes word2vec for the graph domain, where intuitively, each node is regarded as a word. The algorithm creates the equivalent to sentences by performing random graph walks starting from all nodes (i.e., each node sampled in the random walk is a word in the sentence).

One of the key contributions of node2vec is the generalization that differentiates between space and structure. In other words, one can select whether to embed a node based on other nodes that are closer in space (i.e., same cluster) or based on nodes with a similar role in the structured graph. From a graph algorithm viewpoint, this can be explained as selecting whether to perform random walks

with a BFS bias or with a DFS bias. Given a random walk from node u to node v, node2vec formulates this bias strategy by defining two hyper-parameters, p and q, which help adjust the transition probability $\alpha_{pq}(u,x)$ from node u to some node x:

$$\alpha_{pq}(u,x) = \begin{cases} \frac{1}{p} & if \quad d_{ux} = 0 \\ 1 & if \quad d_{ux} = 1 \\ \frac{1}{q} & if \quad d_{ux} = 2 \end{cases}$$

where d_{ux} stands for the distance between node u and node x.

In this way, node2vec can bias the random walk closer or further away from the source node, creating different embedding types. For example, setting $p< q$ biases the random walk to nodes closer to each other. This in turn causes nodes from the same cluster to be embedded closer and nodes from different regions to be embedded further away. Setting $p>q$ biases the random walk to embed nodes of the same graph characteristics (e.g., same role in a social graph) closer together while others are embedded further away. As node2vec only uses the transition probability, by weighting and directing the random walks, the embedding algorithm can be extended for weighted and directed networks as well. Note that in the special case of p=q=1, the algorithm works similarly to DeepWalk [39].

Due to the scalability of node2vec and its network preservation properties, we use it to initialize Q_t. Specifically, we compute the node embeddings for all T graphs $G_{t_1}, \ldots, G_{t_T}$. The outcome of this stage is a $T \times d$ vector per node, where T is the number of time steps and d is the embedding size. Those are used as initial values for Q_t, which will be further optimized for an end task.

3.2 Temporal Node Embedding Alignment

In this section, we discuss in detail the optimization of the rotation matrix R_t. Word2vec (and analogously node2vec) aims to minimize related word embedding distances, but does not guarantee embedding consistency across two distinct trainings. In other words, two different trainings of word2vec on the same text corpus might result in different word representations. For example, consider a simple transformation over word embedding in a two-dimensional space (x-y axis). One could apply a rotation over each embedding by turning the space around a third axis z. The embeddings of the words would change, but the cosine-similarity between any two word embeddings would remain the same. This transformation represents an example of changes that might occur between two different training periods, resulting in identical cosine similarity values between word pairs, but different individual word embeddings. Similarly, in the temporal graph domain, the embeddings of two graphs G_{t_i}, G_{t_j} are performed independently, and therefore it is not guaranteed, even if the graphs are identical over the time points t_i and t_j, that the node embeddings will remain the same. In other words, the coordinate axes of the embedded graph nodes are not guaranteed to align.

When learning a node evolution over time, we need to ensure that the representations across different time points are consistent. We aim to "smooth out" the variation between two different time steps t_i and t_j that originate from different embedding training sessions. This allows us to capture the true temporal graph evolution. Assuming that most nodes have not changed much between t_i and t_j (e.g., j=i+ϵ), we preform an orthogonal transformation between the node embeddings at time t_i and the node embeddings at time t_j. Specifically, we use Orthogonal Procrustes [19, 43, 44], which performs a least-squares approximation of two matrices and has been widely used in computer vision applications [5, 17]. Applied to our problem, let $Q_t \in \mathbb{R}^{d \times |V|}$ be the matrix of node embeddings at time step t. We align the matrices corresponding to consecutive time steps iteratively, i.e., align Q_{t_2} to Q_{t_1}, followed by aligning Q_{t_3} to Q_{t_2} and so forth. The alignment requires finding the orthogonal matrix R between Q_t and Q_{t+1}. The final embedding at time Q_t is now $Q'_t = RQ_t$. The approximation is performed by optimizing the following regression problem:

$$R_{t+1} = \underset{R \text{ s.t. } R^T R = I}{\operatorname{argmin}} \|RQ_{t+1} - Q_t\|$$

其中R为旋转矩阵，简单却非常有效，下图是作者做的任务，也看不出来跟公式的关系，作者想做的事情是不同时刻特征对齐，直观的理解见作者引用的机器翻译的那篇论文上的图。

where $R_{t+1} \in \mathbb{R}^{d \times d}$ is the best orthogonal transformation alignment between the two consecutive time steps. Notice the regression problem is performed on nodes that appear both at time t and time t+1. New nodes that only appeared at t+1 are transformed using R_{t+1}.

3.3 Node Embedding over Time

In eq. 1, the final embedding is dependent on matrices A, B, and an activation function σ, which are jointly optimized as descibed in eq. 2. In this section, we discuss the choice of A, B and σ and the final joint optimization.

Intuitively, one can try to break the optimization process to several steps. The first embeds each node in the graph G (Section 3.1) and the second aligns the graph embeddings over time (Section 3.2). Following these steps, each node, v, is now associated with a matrix $X^{(v)} \in \mathbb{R}^{T \times d}$ of its historical T embeddings over time, each of size d. The graph G is associated with $|V|$ matrices, one for each node: $G_X = X^{(v_1)}, \ldots, X^{(v_{|V|})}$. Given G_X, we wish to perform graph prediction tasks – node classification and link prediction.

The common approach for these tasks is to represent a node via a d-dimensional vector, which can then be used as input to any supervised machine learning-classifier [2, 31, 32]. Similar to our problem but in the text domain, a sentence consists of a set of words, each with an associated embedding of size d. For the task of sentence classification, each sentence is embedded into a a vector of size d, which is then fed into a classifier.

Analogously, in this work, at the final steps of the optimization, we aim to create for each node a single d-dimensional representation, leveraging its T historical embeddings, $X^{(v)}$. In order to reduce sequence data into a d-dimensional vector, we use recurrent neural networks (RNNs) with long short term memory (LSTM) [18]. Figure 1a illustrates the process for the multi-label node classification task. Each node embedding at time t, after alignment, is fed as input to an LSTM memory cell of size d. The last memory cell $h_T \in \mathbb{R}^d$ of the LSTM represents the final temporal embedding of the node, v^t, optimizing for the specific task. For the link prediction task (Figure 1b), each edge $e = (v_1, v_2)$ is represented by (v_1^t, v_2^t) – a vector of size $2d$, which is the concatenation of the two nodes it connects. This edge representation is then fed into a fully-connected (FC)

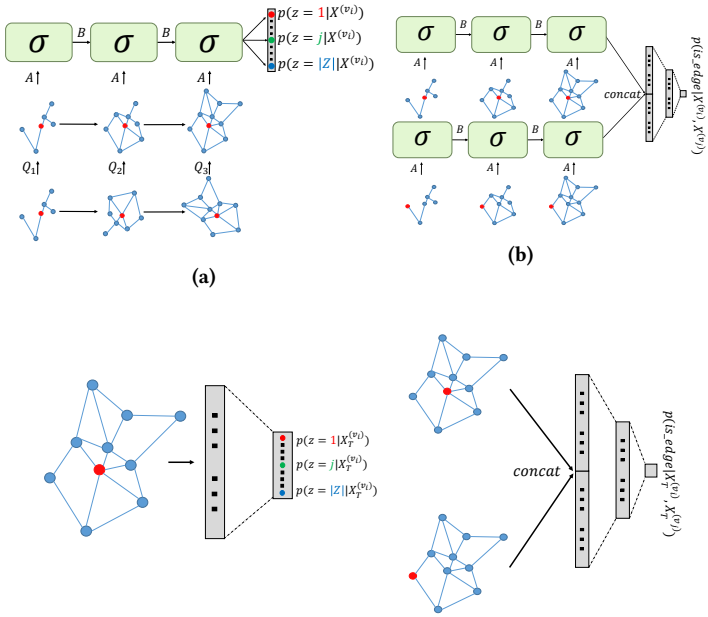

Figure 1: End-to-end architecture for link prediction and node classification. Figure (a) presents the multi-label node classification architecture. The embeddings are initialized using static graph embeddings. Those are then aligned per each time step, and sequentially optimized to capture historical information that might be necessary for the classification. Eventually, a vector of probabilities for each class node is produced. Figure (b) presents the link prediction architecture. The embedding of both nodes connected by the edge, after alignment, are concatenated and fed into a fully-connected layer, followed by a softmax layer, whose output is the link probability. Similarly, Figure (c) presents the static architecture for multi-label node classification (left) and link prediction (right).

layer of size d followed by a softmax layer. Intuitively, The FC layer tries to predict the existence of the potential edge.

4 EXPERIMENTAL SETUP

In this section, we describe our datasets, our two experimental tasks, and our baselines, which include a static embedding and previously-published temporal graph prediction algorithms.

4.1 Datasets

Table 1 lists the different datasets we used for our experiments and their characteristics. Below, we describe each of them in more detail.

arXiv hep-ph. A research publication graph, where each node is an author and a temporal undirected edge represents a common publication with a timestamp of the publication date. Time steps reflect a monthly granularity between March 1992 and December 1999.

Facebook friendships. A graph of the Facebook social network where each node is a user and temporal undirected edges represent users who are friends, with timestamps reflecting the date they became friends. Time steps reflect a monthly granularity between September 2004 and January 2009.

Facebook wall posts. A graph of the Facebook social network where each node is a user and a temporal directed edge represents a post from one user on another user's wall at a given timestamp. Time steps reflect a monthly granularity between September 2006 and January 2009.

CollegeMsg. An online social network at the University of California, with users as nodes and a temporal directed edge representing a private message sent from one user to another at a given timestamp. Time steps reflect a daily granularity between April 15th, 2004 and October 26th, 2004.

PPI. The protein-protein interactions (PPI) graph includes protein as nodes, with edges connecting two proteins for which a biological interaction was observed. Numerous experiments have been conducted to discover such interactions. These are listed in HINTdb [37], with the list of all articles mentioning each interaction. We consider the interaction discovery date as the edge's timestamp. In a pre-processing step, we set it as the earliest publication date of its associated articles. We work in a yearly granularity between 1970 and 2015. We publicly release this new temporal graph.[5]

Slashdot. A graph underlying the Slashdot social news website, with users as nodes and edges representing replies from one user to another at a given timestamp. Time steps reflect a monthly granularity between January 2004 and September 2006.

Cora. A research publication graph, where each node represents a publication, labeled with one of L=10 topical categories: artificial intelligence, data structures algorithms and theory, databases, encryption and compression, hardware and architecture, human computer interaction, information retrieval, networking, operating systems, and programming. Temporal directed edges represent citations from one paper to another, with timestamps of the citing paper's publication date. Time steps reflect a yearly granularity between 1900 and 1999.

DBLP. A co-authorship graph, focused on the Computer Science domain. Each node represents an author and is labeled using conference keywords representing L=15 research fields: verification testing, computer graphics, computer vision, networking, data mining, operating systems, computer-human interaction, software engineering, machine learning, bioinformatics, computing theory, security, information retrieval, computational linguistics, and unknown. Temporal undirected edges represent co-authorship of a paper, with timestamps of the paper's publication date. Time steps reflect a yearly granularity between 1990 and 1998.

Notice that for the arXiv hep-ph, Facebook Wall Posts, CollegeMsg, Slashdot, and DBLP graphs, multiple edges may occur from one node to another at different timestamps. Given a timestamp, if multiple edges exist, they are collapsed into a single edge, weighted according to the number of original edges, thus rendering a weighted graph, as marked in Table 1.

Table 1: Dataset characteristics.

Dataset	Weighted	Directed	Nodes	Edges	Diameter	Train time steps
arXiv hep-ph[1]	+	-	16,959	2,322,259	9	83
Facebook friendships[2]	-	-	63,731	817,035	15	26
Facebook wall posts[3]	+	+	46,952	876,993	18	46
CollegeMsg[4]	+	+	1,899	59,835	8	69
PPI[5]	-	-	16,458	144,033	10	37
Slashdot[6]	+	+	51,083	140,778	17	12
Cora[7]	-	+	12,588	47,675	20	39
DBLP[8]	+	-	416,204	1,436,225	23	9

4.2 Experimental Tasks

We evaluated our temporal node embedding approach with regards to two fundamental tasks: temporal link prediction and multi-label node classification. For link prediction, the first six datasets in Table 1 were used, while for node classification the remaining two datasets, Cora and DBLP, which include node labels, were used.

4.2.1 Temporal Link Prediction. This task aims at forecasting whether an edge will be formed between two nodes in a future time point.

Data: We divided the data into train and test by selecting a pivot time, such that 80% of the edges in the graph (or the closest possible portion) have a timestamp earlier or equal to the pivot. Following, all edges with an earlier (or equal) timestamp than the pivot time were considered as positive examples for the training set, while all edges with a timestamp later than the pivot time were considered as positive examples for the test set. For negative examples in the training set, we sampled an identical number of edges to the positive examples, uniformly at random out of all node pairs not connected at pivot time. For negative examples in the test set, we sampled uniformly at random an identical number of edges to the positive examples, from all node pairs not connected by an edge with any timestamp. We focused our task on predicting ties between existing nodes and therefore restricted edges in the test set to be formed only between existing nodes in the training set.[9]

Metric: As evaluation metric for all methods, we used the area under the ROC curve (AUC).

4.2.2 Multi-Label Node Classification. This task aims at predicting the label of a node out of a given set of L labels at a given time point.

Data: For this task, we randomly split the entire dataset so that 80% of the nodes are used for training and 20% are used as the test set.

Metrics: As our main evaluation metric for all methods, we used the F1 score. We examined both the micro F1 score, which is computed globally based on the true and false predictions, and the macro F1 score, computed per each class and averaged across all classes. For completeness, we also report the AUC.

[1]http://konect.uni-koblenz.de/networks/ca-cit-HepPh
[2]http://konect.uni-koblenz.de/networks/facebook-wosn-links
[3]http://konect.uni-koblenz.de/networks/facebook-wosn-wall
[4]https://snap.stanford.edu/data/CollegeMsg.html
[5]https://github.com/urielsinger/Datasets
[6]http://konect.uni-koblenz.de/networks/slashdot-threads
[7]https://people.cs.umass.edu/~mccallum/data.html
[8]http://dblp.uni-trier.de/xml
[9]The selection of the pivot time took into account this restriction.

4.3 Baselines

In our evaluation for both tasks, we compare tNodeEmbed to the following baselines:

Node2vec [14]: we use node2vec as a static baseline representing the state of the art. It is also the static node embedding algorithm we opted to use for the initialization of tNodeEmbed, therefore the comparison between them is of special interest.[10] For temporal link prediction, edge embedding is implemented by concatenating the two respective node2vec embeddings at pivot time and classification is performed as described in Section 3.3. For node classification, node embeddings are produced at the last time step and are input into an FC layer, which outputs, per each node, a vector of size L representing its class probabilities. The architectures can be seen in Figure 1c.

Temporal Matrix Factorization (TMF) [10]: this method, used specifically for temporal link prediction, collapses the data across multiple time points into a single representation over which matrix factorization is applied. Specifically, given T adjacency matrices $A_1, \ldots, A_T$ representing T snapshots, a collapsed adjacency matrix is created by weighting them as follows: $\sum_{t=1}^{T}(1-\theta)^{T-t}A_t$. This weighting scheme gives higher weight to matrices representing later time points, using a decay factor θ, set to 0.2 [10]. Different factorization methods were proposed on top of the collapsed matrix representation, some of which applicable only to bipartite graphs. For the *TMF* baseline, we examine singular value decomposition (SVD) as a representative factorization technique. In addition, we also experiment with a setting in which node2vec is used on top of the collapsed matrix, as a more modern method for embedding, and mark this variant as TMFntv. For both TMF and TMFntv, the embedded representation of the matrix is fed into the static architecture shown in Figure 1c.

Temporally Factorized Network Modeling (TFNM) [55]: this method applies factorization before collapsing, by generalizing the regular notion of matrix factorization for matrices with a third 'time' dimension. Given T adjacency matrices $A_1, \ldots, A_T$ representing a temporal graph, factorization is performed using a static matrix U and a time-dependent matrix $V(t)$, i.e., $A_t = UV(t)^T$, where $V(t) = W_0 + W_1 t$. This yields a first-order polynomial connection between the A_t matrices. To determine the values of U and $V(t)$, the following optimization is applied:

$$\min_{U,V} J(U,V) = \sum_{t=1}^{T} \frac{D(t)}{2} ||1_{E(t)}(A_t - UV(t)^T)||_F^2$$

Here, $1_{E(T)}$ is the binary (unweighted) version of A_t and $D(t) = e^{-\theta(T-t)}$ is a decay function that gives higher weight to more recent time points, with θ set to 0.3 [55].[11]

For the link prediction task, after U and V are determined via the optimization above, the adjacency matrix at time $T+n$ is determined by calculating $UV(T+n)^T$. For the node classification task, the matrix $V(T)$, in which the i-th row represents the embedding of

[10]For this baseline, we used the implementation published by the authors: https://github.com/aditya-grover/node2vec
[11]For this baseline, we used an implementation kindly shared with us by the authors [55].

the i-th node at the final time step T, is fed into the corresponding static architecture shown in Figure 1c.

Continuous-Time Dynamic Network Embeddings (CTDNE) [34]: this method is based on random walks with the application of a stipulation that the timestamp of the next edge in the walk must be larger than the timestamp of the current edge. This leads to the creation of 'chronological' paths, which represent the order of events. For example, if 'a' talks to 'b' and only then 'b' talks to 'c', a temporal path of the form 'c'->'b'->'a', which is legitimate in a regular random walk (as in the static DeepWalk method [39]), will not be allowed. In addition, the likelihood of a random walk step is also weighted according to the temporal proximity of the edges. Notice that this baseline refers to a weighted edge in its raw form, i.e., multiple equally-weighted edges, each with its own timestamp. For this baseline, we used the finest time granularity included in each dataset, i.e., seconds for all datasets, except for PPI (days), Cora (years), and DBLP (years). The embeddings are then fed into the static architecture shown in Figure 1c.

Hawkes process-based Temporal Network Embedding (HTNE) [59]: this method works similarly to CTDNE, but with neighborhoods generated by modeling Hawkes processes [16], where each edge is weighted exponentially by the time difference. Similar to CTDNE, we used the finest time granularity included in each dataset. The embeddings are then fed into the static architecture shown in Figure 1c.

DynamicTriad (DynTri) [56]: this method uses the modeling of the triadic closure process to learn representation vectors for nodes at different time steps. In particular, it models how a closed triad, which is composed of three nodes connected to one another, develops from an open triad, which has one pair of disconnected nodes. Since this triadic closure process is a fundamental mechanism in the formation and evolution of networks, it enables capturing the network dynamics and learning representation vectors for each node at different time steps. Overall, the loss function for each time step tries to optimize three factors: smoothing the embeddings of the same node in the current and previous time step; the embeddings of two nodes that have an edge between them in the current time step; and the embeddings of nodes that have many common 'close' friends, as the triadic closure is more probable with such friends.[12] The embeddings of the last time step are then fed into the static architecture shown in Figure 1c.

5 EXPERIMENTAL RESULTS

The principal part of our evaluation includes a detailed comparison of tNodeEmbed with all baselines described in the previous section for the temporal link prediction and multi-label node classification tasks. We then further examine the performance for the link prediction task over four types of random graphs, with different degree distributions. Following, we inspect the effect of our alignment step on performance of both the link prediction and node classification tasks. We then examine the sensitivity of tNodeEmbed to different hyper-parameters and analyze the effect of the number of time steps T on its performance. We conclude with an analysis of the time and space complexity of tNodeEmbed.

[12]For this baseline, we used the implementation published by the authors: https://github.com/luckiezhou/DynamicTriad/#dynamictriad

5.1 Temporal Link Prediction

Table 2 presents the performance results of tNodeEmbed compared to the different baselines for the link prediction task. It can be seen that tNodeEmbed outperforms all other baselines across three datasets (Facebook wall posts, PPI, and Slashdot) and reaches comparable results in the other three. It also poses the most consistent performance, achieving the highest result for all six datasets. The performance results as well as the gap from the baselines vary substantially across the datasets. For arXiv hep-ph and Facebook friendships, node2vec and TMFntv (our own composed baseline combining temporal matrix factorization with node2vec embedding) achieve comparable results to tNodeEmbed. For CollegeMsg, both CTDNE and HTNE achieve comparable results to tNodeEmbed. Interestingly, it can be noticed that the static node2vec baseline outperforms TMF and TFNM across all datasets, with the exception of Slashdot for TMF. This indicates the strength of modern node embedding methods and suggests that the temporal data across all time steps is not guaranteed to yield a performance gain on top of such embedding. The poor results of TFNM may stem from the fact we examine substantially larger graphs. It also implies that the assumption of a first-order polynomial tie across time points may be too restrictive in such cases. The particularly low performance demonstrated by DynTri may stem from the fact that the loss in each time step uses only data from the current and previous time steps. As a result, the broader dynamics of the graph are not captured as effectively as in other methods. Indeed, the authors demonstrated the effectiveness of this method for tasks that use the node representations in time t to make predictions for time t or t+1, but not in further steps, where the entire dynamics of the graph become important to capture.

To better understand on which types of graphs tNodeEmbed is most effective, we analyzed several graph properties and discovered a particularly consistent correlation with the global *clustering coefficient* (CC) [51]. This is a network property that measures the ratio between the number of triangles in the graph (multiplied by a factor of 3) and the number of pairs of connected edges. Intuitively, the CC reflects how well nodes in a graph tend to cluster together, or how *cohesive* the graph is. As Table 2 indicates, tNodeEmbed tends to perform better compared to the baselines for graphs with a lower CC. We conjecture the reason lies in the cohesiveness of the graph over time. As most graph generation processes follow the preferential attachment model [3], nodes tend to attach to existing communities, rendering them more cohesive. The "denser" the graph, the easier it is to predict the appearance of a link, as edges tend to attach to higher degree nodes.

5.2 Multi-Label Node Classification

Table 3 presents the results for the node classification task. In this case, tNodeEmbed outperforms all other baselines across all metrics over both the Cora and DBLP datasets, except for the case of micro F_1 for DBLP, which is equal to that of TFNM. However, for the latter, the macro F_1 is especially low, implying a collapse into one label. A similar phenomenon of a low macro F_1 can be observed for CTDNE, HTNE, and for DynTri. This suggests that the time scale of years in both the Cora and DBLP datasets is not suitable for continuous methods. In addition, it reinforces our conjecture

Table 2: AUC performance of tNodeEmbed vs. baselines for the link prediction task. Boldfaced results indicate a statistically significant difference. The clustering coefficient (CC) of each graph is presented at the rightmost column.

Dataset	tNodeEmbed	node2vec	TMF	TMFntv	TFNM	CTDNE	HTNE	DynTri	CC
arXiv hep-ph	0.951	0.948	0.908	0.950	0.738	0.905	0.851	0.783	0.291
Facebook friendships	0.939	0.938	0.886	0.925	0.814	0.757	0.724	0.535	0.148
Facebook wall posts	**0.917**	0.902	0.718	0.900	0.720	0.827	0.784	0.643	0.078
CollegeMsg	0.841	0.823	0.809	0.819	0.654	0.841	0.838	0.630	0.036
PPI	**0.828**	0.799	0.753	0.798	0.712	0.800	0.782	0.761	0.017
Slashdot	**0.913**	0.777	0.896	0.793	0.661	0.894	0.886	0.765	0.010

Table 3: Performance results of tNodeEmbed vs. baselines for the node classification task over the Cora and DBLP datasets. Boldfaced results indicate a statistically significant difference. The clustering coefficient (CC) of each graph is presented in parenthesis.

Algorithm	Cora (CC=0.275)			DBLP (CC=0.002)		
	Micro F_1	Macro F_1	AUC	Micro F_1	Macro F_1	AUC
tNodeEmbed	**0.668**	**0.513**	**0.925**	0.822	**0.504**	**0.977**
node2vec	0.547	0.284	0.862	0.752	0.235	0.943
TMF	0.552	0.361	0.875	0.740	0.203	0.937
TMFntv	0.511	0.246	0.856	0.749	0.219	0.939
TFNM	0.386	0.078	0.760	0.822	0.060	0.937
CTDNE	0.374	0.054	0.753	0.717	0.083	0.916
HTNE	0.391	0.056	0.747	0.714	0.069	0.911
DynTri	0.386	0.055	0.746	0.711	0.055	0.897

that DynTri may work well only for predictions for the current or next time step.

It can also be seen that as opposed to the link prediction task, the performance gaps between tNodeEmbed and the baselines are rather similar for Cora and DBLP, despite the CC difference between the two graphs. Indeed, by contrast to link prediction, node classification is not directly related to the cohesiveness of the graph.

Overall, our evaluation indicates that tNodeEmbed achieves clear performance gains over the baselines for both the link prediction and node classification tasks.

5.3 Degree Distribution

In order to further explore the performance of tNodeEmbed on different types of graphs, and the aforementioned effect of the clustering coefficient, we superficially generated four random graphs with different degree distributions other than power law: linear, logarithmic, sinusoidal, and exponential. All graphs were created for n=1000 nodes and m=100,000 edges. We increased the number of edges linearly along T=50 time steps, so that the total number of edges after all T time steps would be m. At each time step t, we added m_t new random edges to the graph, between its n nodes, so that the degree distribution would be as close as possible to the predefined degree distribution. We ran tNodeEmbed, node2vec, and CTDNE for the temporal link prediction task, using t=45 as the pivot, over this set of four random graphs. We opted for CTDNE as a representative temporal baseline, since it achieved the best performance over 3 of the 6 datasets for the temporal link prediction task (as can be seen in table 2). Notice that since the edge selection is not completely random, but has to follow a predefined degree distribution, node embedding is still expected to be meaningful.

Table 4: AUC performance for temporal link prediction over randomly generated graphs with different degree distribution targets. Boldfaced results represent a statistically significant difference. The clustering coefficient (CC) of each graph is presented at the rightmost column.

Distribution	tNodeEmbed	node2vec	CTDNE	CC
Linear	0.67	0.53	0.65	0.22
Logarithmic	0.74	0.56	0.73	0.26
Sinusoidal	0.79	0.64	0.78	0.31
Exponential	0.85	0.79	0.83	0.36

Table 4 shows the performance results, which are consistent with those observed for real-world graphs with a power-law degree distribution. For all four distributions, tNodeEmbed outperforms node2vec and is comparable with CTDNE. As previously observed, performance increases as the clustering coefficient of the graph grows, while the gap of tNodeEmbed over node2vec is larger when the cluster coefficient is lower. The similar results for tNodeEmbed and CTDNE algorithms imply that the temporal dynamics for these graphs is relatively easy to capture.

5.4 Alignment

As explained in Section 3.2, tNodeEmbed aligns node embeddings over different time points. This alignment aims to learn consistent node behavior over time and reduce noise that arises from training different embeddings. To examine the effect of the alignment step, we experimented with a variant of tNodeEmbed that skips this step. Table 5 presents the results of this variant alongside the results of the "full-fledged" tNodeEmbed for the temporal link prediction task. It can be seen that the removal of the alignment step leads to a decrease in performance in three out of six datasets, while in the rest the performance is comparable. Overall, these results indicate that the alignment step can play a key role in performance enhancement. We observe that the lower the CC of the graph, the higher the contribution of the alignment step. We conjecture that

Table 5: AUC performance for link prediction using tNodeEmbed with and without alignment. Boldfaced results represent statistically significant differences. The clustering coefficient (CC) of each graph is presented at the rightmost column.

Dataset	Alignment	No Alignment	CC
arXiv hep-ph	0.951	0.950	0.29
Facebook friendships	0.939	0.935	0.15
Facebook wall posts	0.917	0.916	0.08
CollegeMsg	**0.841**	0.825	0.04
PPI	**0.828**	0.822	0.02
Slashdot	**0.913**	0.860	0.01

Table 6: Performance results of tNodeEmbed for node classification with and without alignment. Boldfaced results represent statistically significant differences. The clustering coefficient (CC) of each graph is presented at the rightmost column.

Dataset	Micro F_1		Macro F_1		AUC		CC
	with	without	with	without	with	without	
Cora	**0.668**	0.644	**0.513**	0.475	0.925	0.919	0.275
DBLP	**0.822**	0.785	**0.504**	0.390	**0.977**	0.959	0.002

the variability of the embeddings constructed between each embedding trainings is higher for graphs that are less cohesive. This generates more noise in the data, which leads to a lower performance of tNodeEmbed. The alignment step helps reduce this noise by aligning the embeddings across time points. The results for the node classification task follow the same trends and are presented in Table 6.

5.5 Parameter Sensitivity

Our framework includes a variety of hyper-parameters that influence its performance:

- **p** and **q**, which adjust the transition probability between nodes in the random walk, as explained in Section 3.1. The default value is 1 for both p and q.
- **d** – the embedding size of the node's vector. This hyper-parameter originates from word2vec where it indicates the size of the low-dimensional space in which words are represented. Default value is 128.
- **r** – the number of random walks starting from a specific node. This hyper-parameter determines the size of our corpus, as $|V|\times r$ is the number of sentences. Default value is 10.
- **l** – the length of a sentence generated by the random walk. Default value is 80.
- **k** – another hyper-parameter originating from word2vec, which indicates the size of the window used for the skipgram version. All words within distance k from word w are used as the context words for w when learning the embeddings. Default value is 10.

We tuned each hyper-parameter separately, while fixing the others to their default values, and present the results in a similar way

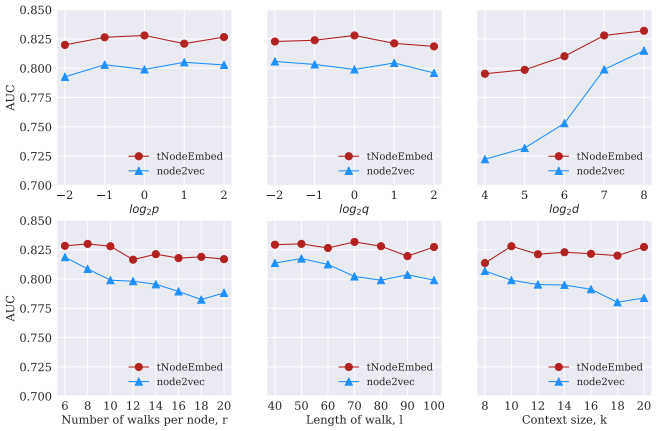

Figure 2: Parameter sensitivity for link prediction over the PPI dataset .

as reported for node2vec [14] in Figure 2. The results are presented for the link prediction task and the PPI dataset, however similar trends were observed for the other datasets and tasks. Overall, the results indicate that tNodeEmbed is less sensitive than node2vec to changes in the d, r, and k hyper-parameters. This reduced sensitivity can be explained by the temporal dimension managing to prevent the feature loss. Looking closely, it can be seen that the least stable hyper-parameter of tNodeEmbed is d, which makes sense since an embedding size that is too small can directly reduce the embedding's effectiveness. Another potential explanation is that the alignment becomes less effective for smaller node representations, which may yield a noisier embedding over the time steps.

5.6 Time Steps

We also set out to explore the effect of the number of time steps on tNodeEmbed performance. To this end, for both the link prediction and node classification tasks, given a dataset with T time steps, we ran tNodeEmbed with $0.1T$, $0.2T$, ..., $0.9T$,T time steps (while, for the link prediction task, keeping the pivot time fixed and analyzing on the train time steps). The last time step was included in all subsets, making sure all the nodes and edges are accounted for.

Figure 3 (Left) shows the results of this analysis for the link prediction task over the collegeMsg, PPI, and Slashdot graphs, while Figure 3 (Right) shows the results of this analysis for the node classification task over Cora and DBLP. As could be expected, the performance of tNodeEmbed consistently increases as the number of time steps grows – the higher the amount of training data used for the optimization, the better its performance. Another interesting observation is that the results obtained with a low number of time steps are similar to the results obtained when not using the alignment method (Section 5.4). As we discussed in Section 3.2, using alignment reduces the noise when trying to learn the node dynamics. Similarly, a low number of time steps yields noisy data that prevents the algorithm from learning the correct dynamic behavior of the node.

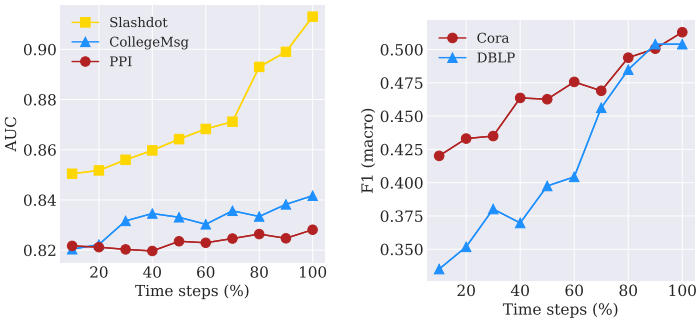

Figure 3: Performance of tNodeEmbed by the portion of time steps. Left: Link prediction. Right: Node classification.

5.7 Time and Space Complexity

Let V be the graph vertices, T the number of time steps and d the embedding size. Node2vec [14] holds a time complexity of $O(|V|^2)$. The method generates $|V|r$ walks each of length k, and then calculates the final embeddings of $|V|d$ parameters. Theretofore, the tNodeEmbed initialization stage has a time complexity of $O(T|V|^2)$, as it performs an initialization step by applying Node2vec over T time steps. tNodeEmbed aligns T time steps in a time complexity of $O(Td^2)$, and the joint loss optimization performs optimization in a time complexity of $O(T|V|d)$. As T and d are small and constant compared to $|V|$, in total, the time complexity of tNodeEmbed is $O(|V|^2)$. The lowest time complexity among the other temporal state-of-the-art methods is of $O(|V|^2)$ [10, 34, 55], while the highest is of $O(|V|^2|E|)$ [56]. We conclude our method is at the lower bound of time complexity compared to the state-of-the-art methods.

As for space, the complexity of node2vec is $O(|V|d)$ [14], henceforth during the initialization of the embeddings using node2vec, the space complexity is $O(|V|dT)$, as it performs the initialization for each time step. During alignment $d^2(T-1)$ rotation matrices are created and an additional d^2 parameters are used during the final optimization. The final space complexity is therefore $O(|V|)$. Most other temporal state-of-the-art methods are of similar space complexity of $O(|V|)$ [10, 34], while some reach a space complexity of $O(|V|^2|)$ [55, 56]. We conclude our method is at the lower bound of space complexity compared to the state-of-the-art methods.

6 CONCLUSIONS

In this paper, we explore temporal graph embeddings. Unlike previous work in predictions over temporal graphs, which focused on optimizing one specific task, we present a framework that allows to jointly optimize the node representations and the task-specific objectives. Our method outperforms a variety of baselines across many of the datasets and does not underperform for any of them. We evaluate our method over a particularly diverse set of large-scale temporal graph datasets, weighted and unweighted, directed and undirected, with different time spans and granularities, for two fundamental graph prediction tasks – node classification and edge prediction.

Our method generalizes graph embedding to capture graph dynamics, somewhat similarly to the extension of word embedding to sequential sentence embedding in natural language processing. Our framework can leverage any static graph embedding technique and learn a temporal node embedding altering it. As graph embeddings do not preserve coherence, i.e., each training of a node's embedding on the same graph can provide a different representation, we explore several approaches for aligning the graph representations to only capture true network dynamics. We then build an end-to-end model that learns a final temporal node embedding, which does not make any assumptions over the temporal behavior (e.g., a polynomial relation between the time points).

As future work, we would like to extend our framework to learn the embeddings by using the best time resolution, without taking snapshots at discrete time points.

REFERENCES

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery* 29, 3 (2015), 626–688.
[2] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. 2006. Link prediction using supervised learning. In *Proc. of SDM06: workshop on link analysis, counter-terrorism and security*.
[3] Albert-Laszlo Barabasi and Reka Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512.
[4] Mikhail Belkin and Partha Niyogi. 2002. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Advances in Neural Information Processing Systems 14*. 585–591.
[5] Paul J Besl and Neil D McKay. 1992. Method for registration of 3-D shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, Vol. 1611. 586–607.
[6] Justin Cheng, Lada Adamic, P. Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. 2014. Can Cascades Be Predicted?. In *Proc. of WWW*. 925–936.
[7] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering* (2018).
[8] Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. 2018. High-Order Graph Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting. *arXiv preprint* abs/1802.07007 (2018).
[9] 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 *IJCAI*. 2086–2092.
[10] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. 2011. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 5, 2 (2011), 10.
[11] Jie Feng, Yong Li, Chao Zhang, Funing Sun, Fanchao Meng, Ang Guo, and Depeng Jin. 2018. DeepMove: Predicting Human Mobility with Attentional Recurrent Networks. In *Proc. of WWW*. 1459–1468.
[12] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.
[13] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2017. DynGEM: Deep Embedding Method for Dynamic Graphs. *arXiv preprint* abs/1805.11273 (2017).
[14] Aditya Grover and Jure Leskovec. 2016. Node2Vec: Scalable Feature Learning for Networks. In *Proc. of KDD*. 855–864.
[15] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint* abs/1709.05584 (2017).
[16] Alan G Hawkes. 1971. Spectra of some self-exciting and mutually exciting point processes. *Biometrika* 58, 1 (1971), 83–90.
[17] Derek LG Hill, Philipp G Batchelor, Mark Holden, and David J Hawkes. 2001. Medical image registration. *Physics in medicine & biology* 46, 3 (2001), R1.
[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[19] John R Hurley and Raymond B Cattell. 1962. The Procrustes program: Producing direct rotation to test a hypothesized factor structure. *Systems Research and Behavioral Science* 7, 2 (1962), 258–262.
[20] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. 2016. Structural-RNN: Deep learning on spatio-temporal graphs. In *Proc. of CVPR*. 5308–5317.
[21] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint* abs/1412.6980 (2014).
[22] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint* abs/1609.02907 (2016).
[23] Andrey Kupavskii, Liudmila Ostroumova, Alexey Umnov, Svyatoslav Usachev, Pavel Serdyukov, Gleb Gusev, and Andrey Kustarev. 2012. Prediction of Retweet Cascade Size over Time. In *Proc. of CIKM*. 2335–2338.

[24] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. 2008. Microscopic Evolution of Social Networks. In *Proc. of KDD*. 462–470.

[25] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *Proc. of KDD*. 177–187.

[26] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed network embedding for learning in a dynamic environment. In *Proc. of CIKM*. 387–396.

[27] Taisong Li, Jiawei Zhang, S Yu Philip, Yan Zhang, and Yonghong Yan. 2018. Deep Dynamic Network Embedding for Link Prediction. *IEEE Access* (2018).

[28] Xiaoyi Li, Nan Du, Hui Li, Kang Li, Jing Gao, and Aidong Zhang. 2014. A Deep Learning Approach to Link Prediction in Dynamic Networks. In *Proc. of SDM*. 289–297.

[29] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint* abs/1511.05493 (2015).

[30] David Liben-Nowell and Jon Kleinberg. 2007. The Link-prediction Problem for Social Networks. *J. Am. Soc. Inf. Sci. Technol.* 58, 7 (May 2007), 1019–1031.

[31] Linyuan Lü and Tao Zhou. 2011. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications* 390, 6 (2011), 1150–1170.

[32] Aditya Krishna Menon and Charles Elkan. 2011. Link prediction via matrix factorization. In *Proc. of ECML PKDD*. 437–452.

[33] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint* abs/1301.3781 (2013).

[34] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Proc. of WWW Companion*. 969–976.

[35] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab K. Ward. 2015. Deep Sentence Embedding Using the Long Short Term Memory Network: Analysis and Application to Information Retrieval. (2015).

[36] Gergely Palla, Imre DerÃĺnyi, IllÃĺs Farkas, and TamÃąs Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (June 2005), 814–818.

[37] Ashwini Patil and Haruki Nakamura. 2005. HINT: a database of annotated protein-protein interactions and their homologs. *Biophysics* 1 (2005), 21–24.

[38] Yulong Pei, Jianpeng Zhang, George HL Fletcher, and Mykola Pechenizkiy. 2016. Node classification in dynamic social networks. In *Proc. of AALTD (ECML/PKDD Workshop)*. 8.

[39] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proc. of KDD*. 701–710.

[40] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. 2004. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences* 101, 9 (2004), 2658.

[41] Ryan A. Rossi and Nesreen K. Ahmed. 2015. Role Discovery in Networks. *IEEE Trans. Knowl. Data Eng.* 27, 4 (2015), 1112–1131.

[42] Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE* 290 (2000), 2323–2326.

[43] Peter H Schönemann. 1966. A generalized solution of the orthogonal procrustes problem. *Psychometrika* 31, 1 (1966), 1–10.

[44] Peter H Schönemann and Robert M Carroll. 1970. Fitting one matrix to another under choice of a central dilation and a rigid motion. *Psychometrika* 35, 2 (1970), 245–255.

[45] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *Proc. of WWW*. 1067–1077.

[46] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. 2000. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science* 290 (2000), 2319.

[47] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. 2017. Know-Evolve: Deep Temporal Reasoning for Dynamic Knowledge Graphs. In *Proc. of ICML*. 3462–3471.

[48] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2018. Representation Learning over Dynamic Graphs. *arXiv preprint* abs/1803.04051 (2018).

[49] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *Proc. of KDD*. 1225–1234.

[50] Hongjian Wang and Zhenhui Li. 2017. Region Representation Learning via Mobility Flow. In *Proc. of CIKM*. 237–246.

[51] Stanley Wasserman and Katherine Faust. 1994. *Social network analysis: Methods and applications*. Vol. 8. Cambridge university press.

[52] Sijie Yan, Yuanjun Xiong, and Dahua Lin. 2018. Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition. In *Proc. of AAAI*. 3482–3489.

[53] Shuicheng Yan, Dong Xu, Benyu Zhang, Hong-Jiang Zhang, Qiang Yang, and Stephen Lin. 2007. Graph Embedding and Extensions: A General Framework for Dimensionality Reduction. *IEEE Trans. Pattern Anal. Mach. Intell.* 29, 1 (2007), 40–51.

[54] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal Graph Convolutional Neural Network: A Deep Learning Framework for Traffic Forecasting. *arXiv preprint* abs/1709.04875 (2017).

[55] Wenchao Yu, Charu C. Aggarwal, and Wei Wang. 2017. Temporally Factorized Network Modeling for Evolutionary Network Analysis. In *Proc. of WSDM (WSDM '17)*. 455–464.

[56] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process. In *Proc. of AAAI*. 571–578.

[57] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. 2018. High-order Proximity Preserved Embedding For Dynamic Networks. *IEEE Transactions on Knowledge and Data Engineering* (2018), 2134–2144.

[58] Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. 2016. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering* 28, 10 (2016), 2765–2777.

[59] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding Temporal Network via Neighborhood Formation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2857–2866.