Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)

Exploiting Interaction Links for Node Classification with Deep Graph Neural Networks

Hogun Park and **Jennifer Neville**
Department of Computer Science, Purdue University
{hogun, neville}@purdue.edu

Abstract

Node classification is an important problem in relational machine learning. However, in scenarios where graph edges represent interactions among the entities (e.g., over time), the majority of current methods either summarize the interaction information into link weights or aggregate the links to produce a static graph. In this paper, we propose a neural network architecture that jointly captures both temporal and static *interaction patterns*, which we call Temporal-Static-Graph-Net (TSGNet). Our key insight is that leveraging both a *static neighbor encoder*, which can learn aggregate neighbor patterns, and a *graph neural network-based recurrent unit*, which can capture complex interaction patterns, improve the performance of node classification. In our experiments on node classification tasks, TSGNet produces significant gains compared to state-of-the-art methods—reducing classification error up to 24% and an average of 10% compared to the best competitor on four real-world networks and one synthetic dataset.

1 Introduction

Node classification is a central task in relational machine learning. In complex network domains, node classification methods have used different types of relational information such as features of direct neighbors [Lu and Getoor, 2003] and autocorrelation of class labels [Jensen *et al.*, 2004]. While these, and other, methods have shown the effectiveness of using neighbor information to improve node classification, many real-world network datasets have sparse link structure which limits the amount of neighbor information available for classification. This is particularly true for networks constructed from interactions over time. Recent work on low-dimensional node embeddings and neural network architectures for graphs [Kipf and Welling, 2016; Grover and Leskovec, 2016] has shown promising results for addressing the sparsity problem in node classification tasks on static graphs. In particular, GCN [Kipf and Welling, 2016] learns individual node embeddings by passing, transforming, and aggregating node feature information across neighbors in an end-to-end fashion.

While many real world social network domains consist of interactions that are changing and evolving over time, it can be difficult to leverage the dynamics of temporal interactions in node classification methods. For example, users develop connections in social networks while interacting and communicating among themselves over time. The temporal patterns of interactions could be potentially useful for predicting their class labels (e.g., political view.) However, when the interaction edges are very sparse in each temporal snapshot or when a node's neighborhood is biased toward a particular class label in different snapshots, it may be difficult to identify temporal patterns that are useful for prediction. In these cases, static aggregated patterns may be more informative. Our proposed model can not only learn temporal interaction patterns, but also model the aggregated neighborhood, and jointly learn how to combine the two views for node classification.

Specifically, in this paper, we propose a novel deep neural network architecture for sequences of interaction graphs, called TSGNet. Our TSGNet model learns a *temporal encoder* that leverages the strengths of both the GCN to discover interaction patterns at each temporal snapshot and a recurrent unit, LSTM [Hochreiter and Schmidhuber, 1997], to capture complex long-term dependencies. To learn the temporal representation more efficiently, we propose a mini-batch training via importance sampling, which reduces the recursive neighborhood expansion across layers and helps to decrease time complexity while maintaining performance. In addition, TSGNet learns a second *neighbor encoder* representation from a static summary of each node's neighborhood. The static and temporal components of the model are jointly estimated to optimize node classification performance. For evaluation, we conduct extensive experiments on both one synthetic and four different real-world networks, with and without attributes, and we observe significant performance gains compared to state-of-the-art methods. Moreover, we conduct a careful ablation study to show that our architecture design is most robust compared to models that use different static and/or temporal components.

2 Problem Definition

2.1 Motivation

Figure 1a-b show examples of interactions in complex networks. Each node and edge indicate an author and a co-

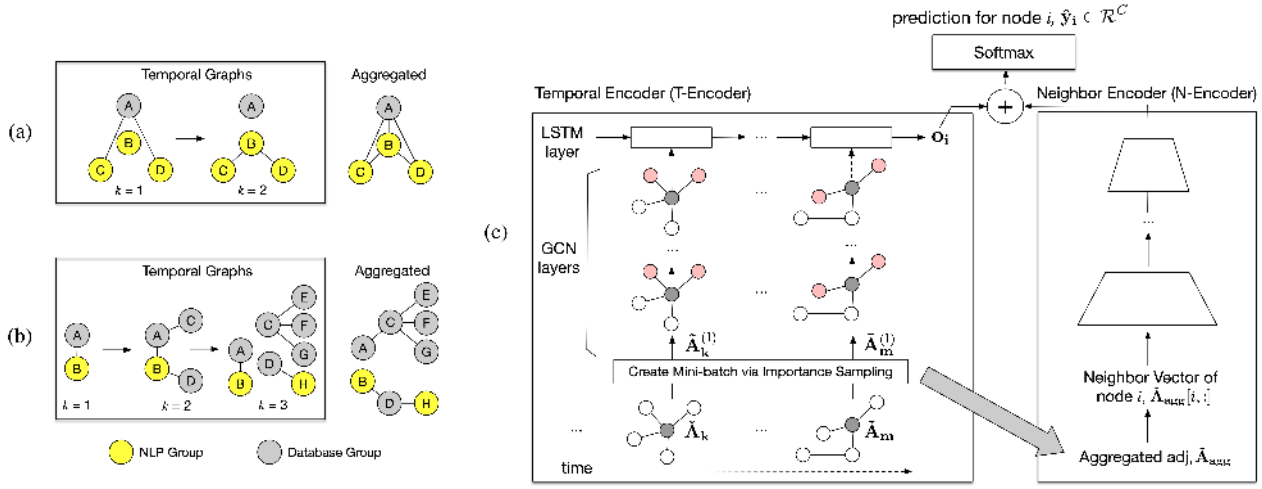


Figure 1: Examples of interactions over time (k): (a) User A and B have same neighbors when aggregated, but different patterns over time. (b) User A and B have same patterns over time but different neighborhood patterns when aggregated. (c) Architecture for *TSGNet.*

author event, respectively. Note that colors represent topic labels, yellow for NLP and grey for Database. In Figure 1a, user A and B have the same coauthors (neighbors) when they are aggregated. In this context, the existing graph embedding approaches (e.g., Node2Vec [Grover and Leskovec, 2016] or GCN [Kipf and Welling, 2016]) will end up learning similar latent representations for the two nodes. However, their temporal interaction patterns may be different. When author A collaborates with other authors differently over time, compared to author B, their representations should be modified accordingly. Meanwhile, if node A and B show similar interaction patterns over time, then it may be difficult to determine the correct class labels through temporal patterns only. In Figure 1b, although the temporal coauthoring patterns around author A and B are similar, their neighborhoods on the aggregated graph are entirely different. In that case, using the aggregated neighborhoods (which correspond to static features) the class labels of node A and B could be identified.

In this paper, our goal is to jointly learn patterns in both interactions over time and in static neighbor sets. In order to model both properties, we propose a neural network model, TSGNet. The details are described in Section 3.

2.2 Notation

We define a graph sequence as a set of graphs such that $G = [G_1, G_2, ..., G_m]$. Each G_k has the same set of nodes, $v_i \in V$ where $\forall i \in [1, n]$, but a different set of edges, $E_k \subseteq V \times V$ such that $G_k = \langle V, E_k \rangle$. If $e_{ij} \in E_k$, there is an edge between v_i and v_j at time k, otherwise there is not. Alternatively, let $A = [\mathbf{A_1}, \mathbf{A_2}, ..., \mathbf{A_m}]$ be the set of adjacency matrices for G, where $\mathbf{A_k}[i, j] = 1$ if $e_{ij} \in E_k$, 0 otherwise. While the network structure is changing over time, we assume that the node attributes are not changing over time[1]. Let F be the feature (attribute) set over the nodes. Each $v_i \in V$ has a corresponding feature vector $\mathbf{f}_i \in F$. Y is the label set over the nodes. Only a subset of the nodes, $v_i \subseteq V$, have a class label, $\mathbf{y_i} \in \mathbb{R}^{|C|}$, where C is a set of class labels. The goal is to learn a model from the partially labeled network and use the model to make predictions $\hat{\mathbf{y}}$ for the unlabeled nodes $\{v_i\}$ $s.t.$ $\mathbf{y_i} \notin Y$. In this work, we assume that Y can be multi-labeled. Moreover, each prediction $\hat{y}_i$ for v_i has an estimated probability.

[1]The setting is realistic for many social and interaction graphs because available node attributes are from basic profiles, resources, or fixed properties, so they are static or changing very slowly. For example, in Facebook, profile attributes like gender and religious views are used as attributes, and in IMDB pre-determined values like contents-rating and budgets are used as attributes of movies.

3 TSGNet for Node Classification

Figure 1c represents the overall architecture for TSGNet. The TSGNet is composed of (1) a static neighbor encoder and (2) multiple layers of GCN for modeling interaction graphs at each time step k. The details of each are described below.

3.1 GCN Layers

We use GCN as a basic component for modeling each temporal graph, $A = [\mathbf{A_1}, \mathbf{A_2}, ..., \mathbf{A_m}]$. Before the data is fed into the GCN, we use the symmetric normalizing trick described in [Kipf and Welling, 2016]. $\mathbf{D_k}$ is the diagonal degree matrix of $\mathbf{A_k} + \mathbf{I}$, and $\mathbf{I}$ is an Identity matrix:

$$\check{\mathbf{A}}_{\mathbf{k}} = \mathbf{D_k}^{-1/2}(\mathbf{A_k} + \mathbf{I})\mathbf{D_k}^{-1/2} \tag{1}$$

Each GCN layer produces node-level output $\mathbf{H_k^{(l+1)}} \in R^{|V| \times Z^{(l)}}$ where $|V|$ is the number of nodes and $Z^{(l)}$ is the size of output representation per a node, which is determined by $\mathbf{W_k^{(l)}}$. The outputs are generated at each time step, k.

$$\begin{aligned} \mathbf{H_k^{(l+1)}} &= f(\mathbf{H_k^{(l)}}, \check{\mathbf{A}}_{\mathbf{k}}) \\ &= (\text{ReLU}(\check{\mathbf{A}}_{\mathbf{k}}\mathbf{H_k^{(l)}})\mathbf{W_k^{(l)}})) \end{aligned} \tag{2}$$

Note that each GCN layer has its own $\mathbf{W_k^{(l)}}$ at each time stamp, and ReLU is used only in the first GCN layer. Moreover, GCN originally uses the attribute matrix for $\mathbf{H_k^{(1)}}$ [Kipf and Welling, 2016]. In this paper, we also support a non-attribute option, where the identity matrix is used for $\mathbf{H_k^{(1)}}$.

Let L be the number of GCN layers for each time step, then the final output for each time step will be an input for an LSTM cell. I.e., $\mathbf{H_k^{(L+1)}}$ for each time step, k, is the temporal input for the k^{th} cell in the LSTM sequence. In Eq. (3) below, $\mathbf{o_i}$ returns the final output vector for v_i in the LSTM. The output, $\mathbf{o_i}$, is projected a final time using the weight matrix, $\mathbf{W}_{\text{lstm}}$, and bias vector, $\mathbf{b}_{\text{lstm}}$. This will be added to the neighbor encoder for v_i below. If the interaction of v_i ends before the last step, m, it still uses the same $\mathbf{W}_{\text{lstm}}$, and $\mathbf{b}_{\text{lstm}}$ to generate the output, $\mathbf{o'_i}$.

$$\begin{aligned} \mathbf{o_i} &= \text{LSTM}(\mathbf{H}^{(\mathbf{L+1})}_{\mathbf{1},...,\mathbf{m}}(v_i)) \\ \mathbf{o'_i} &= \text{softmax}(\mathbf{W}_{\text{lstm}}\mathbf{o_i} + \mathbf{b}_{\text{lstm}}) \end{aligned} \tag{3}$$

3.2 Neighbor Encoder (NE)

The neighbor encoder uses an aggregated matrix $\check{\mathbf{A}}_{\mathbf{agg}}$ as input. $\check{\mathbf{A}}_{\mathbf{agg}}$ is created by aggregating all elements in $[\mathbf{A_1}, \mathbf{A_2}, ..., \mathbf{A_m}]$ and normalizing in the same way described above. The static component reduces the dimensionality for node v_i from its neighbor vector in $\check{\mathbf{A}}_{\mathbf{agg}}$, using stacked fully-connected layers.

$$\begin{aligned} \mathbf{NE_i^{(2)}} &= \text{ReLU}(\mathbf{W^{(1)}}(\check{\mathbf{A}}_{\mathbf{agg}}[i,:] \odot \mathbf{h_i}) + \mathbf{b^{(1)}}) \\ &= ... \\ \mathbf{NE_i^{(L'+1)}} &= \text{softmax}(\mathbf{W^{(L')}}\mathbf{NE_i^{(L')}} + \mathbf{b^{(L')}}) \end{aligned} \tag{4}$$

where $\odot$ refers to the Hadamard product, $\mathbf{h_i} = \{h_{i,j}\}_{j=1}^{|V|}$. Here, if $\check{\mathbf{A}}_{\mathbf{agg}}[i,j] > 0$, $h_{i,j} = \beta$ (for $\beta \geq 1$). Otherwise, $h_{i,j}$ will be equal to 0. This Hadamard product is used to overcome sparsity in the adjacency matrix. If β is 1, it is the same as the adjacency matrix. Otherwise, it puts more weight on non-zero elements in the matrix. It is expected that β will make larger outputs and offset issues from sparsity. In experiments, we set $\beta = 20$, and using this we observe up to 4% of improvement in accuracy.

3.3 Addition Layer

The element-wise addition layer combines the outputs from the GCN-LSTM and the Neighbor Encoder. Specifically, we compute $\hat{\mathbf{v}}_\mathbf{i}$ as the element-wise addition of the outputs from the GCN-LSTM (Eq. 3) and the Neighbor Encoder (Eq. 4):

$$\hat{\mathbf{v}}_\mathbf{i} = \mathbf{NE_i^{(L'+1)}} + \mathbf{o'_i} \tag{5}$$

The addition layer enables a joint representation learned from both the static and temporal neighborhoods around the node. An additional benefit is that the addition layer does not introduce extra parameters, nor does it increase computational complexity. Then, the output, $\hat{\mathbf{v}}_\mathbf{i}$ is put though another softmax layer for classification:

$$\hat{\mathbf{y}}_\mathbf{i} = \text{softmax}(\mathbf{W}_{\text{final}}\hat{\mathbf{v}}_\mathbf{i} + \mathbf{b}_{\text{final}}) \tag{6}$$

Algorithm 1 TSGNet's mini-batched training (one epoch)

```
Generate a mini-batch set B from V
for each mini-batch ∈ B do
    Sample |S| vertices, v_1, ..., v_s ∈ B according to distribution q from mini-batch
    Assign H_k^(1) = H_k^(1)[S, :]
    For l = [1, L−1], assign Ã_k^(l) = Ă_k[S, S]
    Assign Ã_k^(L) = Ă_k[:, S]
    Initialize Ã_agg to |V| × |V| matrix of zeros
    for each v ∈ S do
        Assign Ã_agg[:, v] = Ă_agg[:, v]
    end for
    Compute the categorical cross-entropy in Eq. (7)
    Update W_k^(l), W_lstm, W^(1),...,(L'), b^(1),...,(L'), W_final, and b_final
end for
```

Here, $\hat{\mathbf{y}}_\mathbf{i}$ is the vector output of the softmax function, and each dimension $\hat{y}_{i,j}$ represents the predicted probability of the corresponding class j, given the inputs. For learning, we use categorical cross-entropy (over V_L, the set of labeled nodes) as a loss function at the final layer

$$L = -\sum_{i \in V_L} \sum_{j}^{|C|} y_{i,j} log(\hat{y}_{i,j}) \tag{7}$$

Since all activation functions are differentiable, learning is simply done via back-propagation.

3.4 Importance Sampling

Note that, in Eq. (2), the neighbor aggregation for node u is computed as:

$$(\check{\mathbf{A}}_\mathbf{k}\mathbf{H_k^{(l)}})_u = |V| \sum_{v=1}^{|V|} \frac{1}{|V|} \check{\mathbf{A}}_\mathbf{k}[u,v]\mathbf{H_k^{(l)}}[v,:] \tag{8}$$

which involves a sum over all other nodes in the graph. When we use multiple GCN layers, the recursive neighborhood expansion across layers poses time and memory challenges for training with large graphs. To overcome this limitation, we propose a method for efficient sampling-based learning.

Similar to [Chen *et al.*, 2018], we approximate the equation above with importance sampling. First, we sample a set of S nodes using an importance distribution based on the overall number of interactions, $q(v) = ||\check{\mathbf{A}}_{\mathbf{agg}}[:,v]||^2 / \sum_{v' \in V} ||\check{\mathbf{A}}_{\mathbf{agg}}[:,v']||^2$. Then given the sampled set S, we set $\tilde{\mathbf{A}}_\mathbf{k}^{(\mathbf{l})} = \check{\mathbf{A}}_\mathbf{k}[S,S] \in \mathbb{R}^{|S| \times |S|}$ for all layers $l < L$ and $\tilde{\mathbf{A}}_\mathbf{k}^{(\mathbf{l})} = \check{\mathbf{A}}_\mathbf{k}[:,S] \in \mathbb{R}^{|V| \times |S|}$ when $l = L$, i.e., the last GCN layer. At the last layer, GCN still returns the node representation for all V. Finally, we approximate Eq. (8) for node u as follows:

$$(\check{\mathbf{A}}_\mathbf{k}\mathbf{H_k^{(l)}})_u \approx \frac{|V|}{|S|} \sum_{v=1}^{|S|} \frac{1}{q(v)} \tilde{\mathbf{A}}_\mathbf{k}^{(\mathbf{l})}[u,v]\mathbf{H_k^{(l)}}[v,:] \tag{9}$$

Note the distribution q is only calculated once (i.e., before training) given the normalized aggregated graph and the input feature matrix, $\mathbf{H_k^{(1)}}$, should also be updated according to S via $\mathbf{H_k^{(1)}} = \mathbf{H_k^{(1)}}[S, :]$.

Our overall mini-batched training procedure is described in Algorithm 1. At every epoch, all nodes are randomly divided to create a mini-batch set, B, which is composed of multiples of γ nodes. We set $\gamma = 1024$. B provides a candidate node set for sampling $|S|$ later. When it comes to a new *mini-batch*, $\tilde{\mathbf{A}}_{\mathbf{k}}^{(\mathbf{1})\cdot\cdot(\mathbf{L})}$ is induced from $\check{\mathbf{A}}_{\mathbf{k}}$ according to the S. Similarity, the input matrix of neighbor encoder, $\check{\mathbf{A}}_{\mathbf{agg}}$, is replaced by $\tilde{\mathbf{A}}_{\mathbf{agg}}$. The inner for-loop for $\check{\mathbf{A}}_{\mathbf{agg}}$ retains only the edges in S, but maintains the dimensionality of the NE.

3.5 Complexity Analysis

Recall that m is the number of temporal GCNs and $|F|$ is the number of node attributes. Then assuming the number of hidden units in LSTM, GCN, and NE is constant, the computational complexity of TSGNet is $O(|V| + m|F||E|)$, where $O(|V|)$ is from neighbor encoder and $O(m|F||E|)$ is from the set of temporal GCNs. Because m and $|F|$ are typically much smaller than $|E|$, the time complexity is linear in the number of edges, i.e., $O(|E|)$. With importance sampling, the complexity becomes $O(|V| + |E_S|)$, where $|E_S|$ is the number of edges induced in S. When the non-attribute option is chosen, the complexity is still $O(|V| + |E_S|)$ because the input identity matrix is sparse, thus $|F| = 1$ in the sparse representation, and can be considered as a constant with sparse-dense matrix multiplication.

4 Related Work

Supervised node classification. There is some work on relational models that consider temporal patterns to improve node classification. TVRC [Sharan and Neville, 2008] attempts to model temporal structures through a two-step process. The key idea behind the TVRC is to model the temporal patterns through an exponential weight decay kernel, where the implicit assumption is that network structure in recent past is more important than the structure in the earlier past. The work was extended to ensemble model in [Rossi and Neville, 2012], but it is still limited in learning complex temporal interaction patterns because it heavily relies on the graph summarization with kernel-based edge weighting. This limits the models ability to learn more diverse temporal interactions, and temporal information is lost when collapsing graphs. In addition, DDRC [Park *et al.*, 2017] proposed a convolutional neural network architecture with max pooling for node classification, which models temporal interactions among a node's neighbors. DDRC shows stable performance in spite of different variability of neighbor vectors. However, its effectiveness was partially shown in long and relatively denser graph sequences. Our TSGNet is evaluated from more diverse and larger graph datasets and shows better performance.

Dynamic node embedding. Recently, dynamic network embedding approaches were proposed by [Zhang *et al.*, 2018; Zhu *et al.*, 2018; Ma *et al.*, 2018], which use spectral updates over time for general relational tasks. However, they are evaluated in a synthetic setting, where two temporal snapshots are created by assigning a random timestamp to each edge. Moreover, attributes are also not exploited for learning temporal representations. In addition, CTDNE [Nguyen *et al.*, 2018] learns node representations using temporal random walks. While the method shows promising results on link prediction tasks, it is still limited for learning attributes of nodes and is under-explored to supervised node classification tasks. DynamicTriad [Zhou *et al.*, 2018] also attempted to learn node evolution through representation learning for general relational learning tasks using multiple temporal graph snapshots, but their effectiveness on node classification is still not clear. For example, they are limited to specific types of evolution strategies, and attributes are also not used.

	No Attributes	Static Node Attrs.
Dynamic Edges	**TSGNet**, DynamicTriad	**TSGNet**, DDRC
Static Edges	Node2Vec, LR, NN	NN, GraphSAGE, GCN, ASNE, LR
No Graph	*not applicable*	LR, NN

Table 1: Models categorized w.r.t., the types of relational inputs.

Model categorization. Table 1 categorizes TSGNet and other baseline models according to the types of relational information they use as inputs to their models (w.r.t. edges and attributes). TSGNet uses dynamic interaction edges with and without attributes. In our experiments, we compare TSGNet to all the listed models. Logistic Regression (LR) and Multi-layered Neural Network (NN), Node2Vec [Grover and Leskovec, 2016], and an attributed node embedding, ASNE [Liao *et al.*, 2018], are also employed to model static edges. See Section 5 for more detail. The colors in the table will be used later in the experiments to highlight the performance achieved using each type of relational input.

5 Experimental Evaluation

5.1 Data

We use four real-world network datasets for evaluation. Table 2 reports brief statistics for each network.

Facebook. The Facebook network was scraped from the Purdue University network [Pfeiffer *et al.*, 2015]. Each user (node) is associated with political views for their class labels. An edge is formed when a user writes a post to his or her friend's wall. Users who post more than once a week for at least 8 weeks are chosen. A time window is defined as two weeks. Node attributes are religious views and gender.

	$\|V\|$	$\|E\|$	m	$\|F\|$	$\|C\|$
Facebook	2,716	22,712	55	2	2
DBLP	17,191	318,735	18	2,997	2
IMDB_G	5,043	43,494	65	73	2
IMDB_R	92,611	472,630	14	-	2

Table 2: Network data statistics. m is the number of time windows, and other notations are from Section 2.2.

DBLP. For this dataset, we extract a co-authorship network from the papers published in DBLP from 2000-2017. An edge is created to link two authors when they publish a paper together, with a time stamp based on the publication. Thus, nodes represent authors. Publication venues are selected from AI/NLP and DB conferences[2]. Authors are selected when they have at least 7 years of publication history, and their class label is assigned as the area in which they publish the majority of their papers. Node attributes correspond to term vectors from the titles of each user's published papers.

IMDB_G (Gross Income). We use Kaggle's IMDB (Internet Movie Database) 5,000 movie dataset. An edge is formed when two movies share an actor or actress at each year. All movies have at least 2 temporal edges. A movie has a positive label if the movie gross is larger than 10 million dollars. For this work, we choose budgets, content rating, the number of faces in a movie poster, and genres as features. The budgets are quantized from 0 to 9 using percentiles. Each feature is transformed into one-hot encoding representation.

IMDB_R (Rating). This dataset is from the whole IMDB database[3], and all participants including actors and writers are imported. An edge is formed when two movies share any crew at each year. When a movie's rating is larger than 7.0, it is chosen as a positive label. The periods of all movies are from 2005 and 2018. There are many missing values when all movies are considered, so node attributes are ignored in this dataset. All movies have at least 11 temporal edges.

5.2 Comparison Models

Logistic regression (LR). Logistic regression is performed using neighbor vectors with L1 regularization. This allows us to compare how relational and temporal patterns improve performance. The aggregated (binary or degree normalized (weighted)) graph of all temporal graphs is used for training.

LSTM. We use the TSGNet's input representation for the LSTM, but the GCN layers and the *neighbor encoder* are not used in the architecture. For inputs, the first-hop neighbors at each time window are fed directly into the LSTM layer.

GCN and GraphSAGE. To compare TSGNet with Graph neural networks, GCN [Kipf and Welling, 2016] and GraphSAGE [Hamilton *et al.*, 2017] are evaluated with the aggregated (binary) static graph input. Attributes are used in the same way with TSGNet. We used an LSTM aggregator for GraphSAGE. Because other aggregators for the GraphSAGE such as GCN, mean, and pool are worse than LSTM, the results are not reported here.

Node2Vec. For learning a static node embedding method, Node2Vec, we set $d = [16, 32, 64]$, $r = 10$, $l = 80$, $k = 10$, and p and q were searched over [0.5, 1, 2]. The aggregated (binary) matrix is used for its training.

ASNE. ASNE [Liao *et al.*, 2018] is a recent attributed node embedding method. We used the same hyper-parameter search criteria as in [Liao *et al.*, 2018].

[2]AI/NLP: IJCAI, AAAI, SIGIR, ECIR, CLEF, CHIIR, AIRS, ACL, EMNLP, and COLING; DB: ICDE, VLDB, SIGMOD/PODS, and EDBT.

[3]The IMDB dataset was downloaded November 2018.

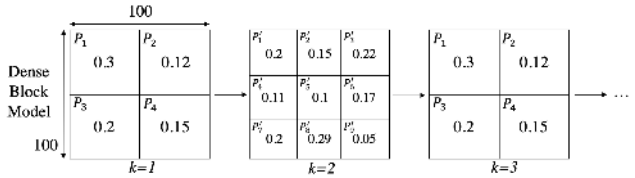


Figure 2: Dense Block Model used to generate synthetic networks. Sparse Block model is 10 times sparser than the Dense Block Model.

DDRC. DDRC [Park *et al.*, 2017] is a CNN-baed temporal classifier, which considers interactions over time. This does not have a *neighbor encoder* or GCN component. The inputs are used as in LSTM above.

Multi-layered neural network (NN). For NN, the *neighbor encoder* of TSGNet is used for training and testing.

TempGCN (GCN+LSTM). This is a version of the TSGNet without the *neighbor encoder (NE)* component where we use GCN to model the temporal graphs with an LSTM.

DynamicTriad. A dynamic node embedding method, DynamicTriad, is also tested with all combinations of parameters as in [Zhou *et al.*, 2018]. Both unweighted graphs and weighted graphs were used for learning, and edges are weighted by the number of common neighbors.

5.3 Evaluation Methodology

Every result we report is the average of 10 trials using randomly shuffled node sets. Note that the entire graph is known before learning, and 70%, 20%, and 10% of node labels are used for training, testing, and validation, respectively. If the accuracy on the validation dataset does not increase during five epochs, learning stops. We also use dropout regularization (0.2) and rectified linear units for activation functions. For optimization, we use the *adam* optimizer [Kingma and Ba, 2014] to update variables. For TSGNet, LSTM, and GCN, the number of hidden nodes is searched over [16, 32, 64, 128], and the numbers of hidden nodes in the neighbor encoder are 512, 128, and 16 at each layer. In addition, there are three GCN layers for TSGNet. For importance sampling, the sampling size, $|S|$ is chosen from [16, 32, 128, 256, 512].

5.4 Results: Synthetic Data

To evaluate the concept of TSGNet, we generate synthetic data from two simplified-Dynamic Stochastic Block Models (DSBM) [Yang *et al.*, 2011] to evaluate our model. We set the number of users to 100, and the length of time-windows for each node is determined by $25 + \text{uniform}(0, 1) \times 25$. The first DSBM (Dense Block Model) is composed of 4 different partitions, P_1, .., P_4, at time$(k)\%2 == 1$. Each partition is composed of 50×50 nodes. In all other time-windows, edges are generated from 9 partitions, P'_1, .., P'_9. Each partition has different edge probabilities, as in Figure 2. The second model (Sparse Block Model) is designed to generate sparse DSBM with low probability. All other conditions are same, but each probability is 10 times sparser than the dense block model. For class labels, 0 is assigned for the first half of nodes (thus, senders of P_1 and P_2), and 1 is set to the second half.

	Dense Block Model	Sparse Block Model
TSGNet	0.89 (0.0299)	**0.884 (0.0282)**
DynamicTriad	**1.0 (0.0)**	0.615 (0.0316)
DDRC	**1.0 (0.0)**	0.705 (0.026)
LSTM	**1.0 (0.0)**	0.530 (0.0376)
GCN	0.519 (0.0272)	0.504 (0.0277)
Node2Vec	0.494 (0.0360)	0.771 (0.0260)
LR	0.429 (0.0164)	0.63 (0.2270)
NN	0.485 (0.0212)	0.688 (0.0326)

Table 3: Classification accuracy on synthetic data. Values in ( ) denote standard errors. Colors indicate type of relational input used by the model from Table 1.

Node classification accuracy (and standard errors) on the synthetic data are shown in Table 3. Bold numbers indicate statistically significant top results. For data from both sparse and dense block models, our proposed model exhibited good performance. While DynamicTriad, DDRC, and LSTM show better performance than TSGNet in the dense data, they are worse in the sparse block model. Meanwhile, other classifiers (GCN, LR, and NN), which were originally designed for static graphs, showed the worse performance than TSGNet's results. Node2Vec works well for the sparse data, but it is worse than TSGNet (p-value < 0.05 in paired t-test).

5.5 Results: Real-world Data

Performance without Node Attributes

Table 4 shows the classification results for the four different real-world datasets. Note that bolded numbers indicate statistically significant top results. (Weighted) in LR refers to versions where the input matrices of the corresponding methods are normalized by the number of edges per each node. In the experiment, TSGNet exhibited the best performance over other alternatives for all datasets and shows comparable performance to TSGNet w/o IS (Importance Sampling). While simple static classifiers such as LR and NN return good performance for Facebook (FB) and DBLP due to the high correlation between neighbor vectors and class labels, however, they are still worse than our TSGNet. These characteristics make TempGCN more difficult to model the data because it is too complex to learn the simple neighborhood. Despite that, the neighbor encoder component of the TSGNet helps it learn the hidden dependencies among nodes and their static neighborhood well. As a result, it produces a significant gain in performance. DDRC and LSTM showed poor performance because the data is also very sparse. DynamicTriads are better than GCN and GraphSAGE in IMDB_G but still worse than TempGCN and TSGNet. Overall, TSGNet produces an average reduction in classification error of 16%, compared to GraphSAGE, which is the best competitor.

Figure 3 shows learning curves on the four datasets as we vary the amount of training data. The learning curves compare the performance as the number of training nodes increases. Note that the set of nodes for testing and validation is same across all range of x-axis. Although the number of training nodes was controlled to calculate the supervised loss, the complete adjacency matrices at each time step for the GCN layers were fixed for the experiment. The experimental assumption was also applied to all other alternatives, TempGCN, Node2Vec, and LR. For Node2Vec, the compete network structure is known for learning representation, and the number of nodes is controlled when its supervised classifier is trained. Therefore, all results with the small training data were not poor. In the Facebook and DBLP datasets, TSGNet was consistently better than the others. For IMDB_G dataset, TSGNet improved in performance as the size of training set increased.

Performance with Node Attributes

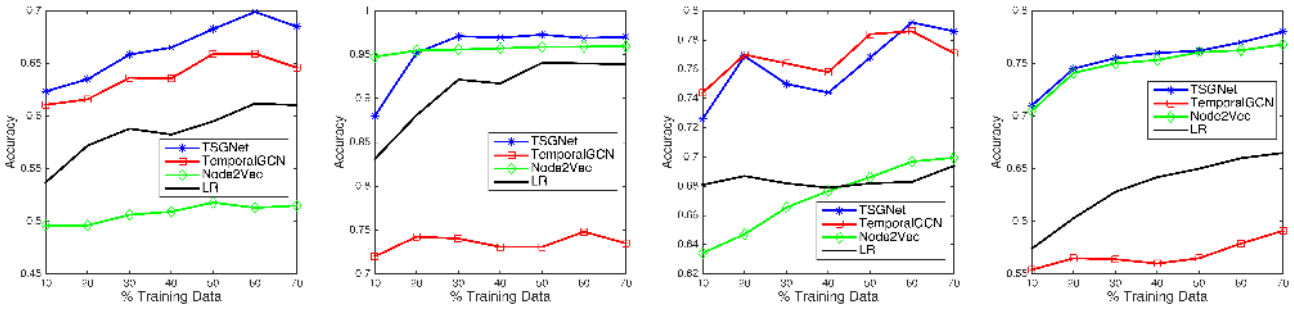
Table 5 shows classification results when node attributes are incorporated into the models. The result for TSGNet with attributes was better than the other alternatives which used attributes in their input. Moreover, the performance of TSGNet without attributes was even better than the result of the best model which uses attributes. Note that, for the DDRC without attributes, an identity matrix is concatenated to the input neighbor vector. This result indicates that it can learn a good representation with only the structural interactions. (attr. + neighbor) refers to a concatenated input including both attribute and neighborhood vectors. ASNE, LR, and NN with the new input show good results in general, but they are worse than TSGNet. ASNE performed poorly on DBLP because it could not utilize labels to learn the embedding. Also, GCN did not work well both with and without attributes. GCN is based on a 1-layer perceptron, which is not a universal approximator [Hornik, 1991]. The 1-layer perceptron in the GCN works like a linear mapping, so the layers may degenerate into simply summing over neighborhood features [Xu *et al.*, 2019]. With this reason, GraphSAGE with LSTM aggregator can model interaction better than GCN for Facebook and DBLP. Overall, TSGNet with or w/o attributes reduces classification error up to 24% and produces an average reduction in classification error of 10%, compared to GraphSAGE.

Temporal Sequence Randomization: Impact on Performance

To see the effect of temporal sequence's randomization, the time-windows were randomly shuffled and used for training. The order of words in language models for NLP and speech recognition is quite important to represent sentences, but the temporal order of social interactions could be reversed

	FB	DBLP	IMDB_G	IMDB_R
TSGNet	**0.68**	**0.97**	**0.78**	**0.78**
TSGNet w/o IS	**0.688**	**0.97**	**0.786**	0.771
TempGCN	0.646	0.734	0.77	0.591
DynamicTriad_W	0.542	0.652	0.732	0.657
DynamicTriad	0.534	0.633	0.730	0.645
DDRC	0.554	0.542	0.717	-
LSTM	0.514	0.538	0.696	-
GraphSAGE	0.645	0.963	0.712	0.752
GCN	0.521	0.665	0.719	0.568
Node2Vec	0.515	0.96	0.7	0.768
NN	0.623	0.83	0.716	0.726
LR	0.593	0.939	0.699	0.665
LR_W	0.613	0.955	0.689	0.673

Table 4: Classification accuracy on real-world datasets. Colors indicate relational input type from Table 1. Results of DDRC and LSTM for IMDB_R are ignored due to the learning time limit ($\geq$1 day).



(a) Facebook (b) DBLP (c) IMDB_G (d) IMDB_R

Figure 3: Learning curves for each dataset as amount of training data is varied.

and often spontaneously happen. In this case, the randomized temporal sequences are likely to represent another instance of evolution. As can be seen in Table 6, TSGNet and TempGCN also work well given the randomized inputs and are not significantly different from the results of original inputs. These results may be interpreted in the context of recent work on Janossy pooling [Murphy *et al.*, 2019] for learning permutation invariant functions with LSTMs via randomization. The fact that the randomized inputs work well within our LSTM architecture may indicate that the model is learning a temporally-invariant function over the interactions.

Ablation Study of Model Components

TSGNet uses GCNs for learning temporal interactions and a NN neighbor encoder for learning the aggregated static first-neighbors. However, we could have chosen other architectures for either component. Table 7 shows the results for different variants of the architecture, with the original components of TSGNet in the first row. Note that we did not use importance sampling to see the true effect of each component. Instead of the GCN in TSGNet, when we use regular densely-connected NN, its performances decreases in DBLP, as shown in the second row of the table. When the GCN is missing in TSGNet like the last row of the table, it also does not work well. Similarly, when the NN in TSGNet is replaced with GCN layers or an empty layer, we can observe the significant drop in Facebook and DBLP. This indicates that our NN-based neighbor encoder helps to jointly learn the temporal network's interaction well if we use GCN layers.

	FB	DBLP	IMDB_G
with Static Attributes			
TSGNet	**0.675**	**0.96**	**0.777**
DDRC	0.554	0.938	0.749
GraphSAGE	0.655	**0.967**	0.717
GCN	0.483	0.881	0.720
ASNE	0.525	0.601	0.734
LR (attr. + neighbor)	0.664	0.96	0.744
NN (attr. + neighbor)	0.645	0.955	0.759
LR (attr. only)	0.63	0.891	0.756
NN (attr. only)	0.63	0.886	0.735

Table 5: Classification accuracy on real-world datasets with node attributes. Colors indicate relational input type from Table 1.

	FB	DBLP	IMDB_G	IMDB_R
TSGNet	**0.688**	**0.97**	**0.786**	**0.78**
TSGNet (R)	**0.679**	**0.96**	**0.774**	0.772
TempGCN	0.646	0.734	**0.771**	0.591
TempGCN (R)	0.658	0.735	0.750	0.573
DDRC	0.554	0.542	0.717	-
DDRC (R)	0.573	0.54	0.718	-
LSTM	0.514	0.538	0.696	-
LSTM (R)	0.480	0.53	0.693	-

Table 6: Classification accuracy with different temporal inputs. (R) denotes that the sequence of inputs is randomized.

N-En	T-En	FB	DBLP	IMDB_G	IMDB_R
NN	GCN	**0.688**	**0.97**	**0.786**	**0.771**
NN	NN	**0.676**	0.953	**0.776**	0.711
GCN	GCN	0.672	0.652	**0.788**	0.707
–	GCN	0.646	0.734	0.771	0.591
–	NN	0.647	0.732	0.769	0.707
GCN	–	0.521	0.665	0.719	0.658
NN	–	0.623	0.83	0.716	0.726

Table 7: Effect of joint learning with different approaches used for the *neighbor encoder* (N-En) and the *temporal encoder* (T-En).

6 Conclusions

In this paper, we described TSGNet, a neural network architecture that can learn jointly from static and temporal neighborhood structure. The architecture exploits the interactions among local neighbors over time, by learning the temporal evolution of a low-dimensional embedding from a GCN, and models its static neighborhood with a densely connected NN. TSGNet is able to improve classification performance by utilizing both patterns in social interactions over time and the set of nodes in the aggregate relational neighborhood.

Acknowledgments

We thank the anonymous reviewers for their useful comments. This research is supported by NSF and AFRL under contract numbers: IIS-1546488, IIS-1618690, and FA8650-18-2-7879. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

References

[Chen *et al.*, 2018] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.

[Hamilton *et al.*, 2017] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of Conference on Neural Information Processing Systems (NeurIPS)*, pages 1024–1034, 2017.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[Hornik, 1991] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[Jensen *et al.*, 2004] David Jensen, Jennifer Neville, and Brian Gallagher. Why collective inference improves relational classification. In *Proceedings of SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 593–598, 2004.

[Kingma and Ba, 2014] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2014.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2016.

[Liao *et al.*, 2018] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2257–2270, 2018.

[Lu and Getoor, 2003] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 496–503, 2003.

[Ma *et al.*, 2018] Jianxin Ma, Peng Cui, and Wenwu Zhu. Depthlgp: learning embeddings of out-of-sample nodes in dynamic networks. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2018.

[Murphy *et al.*, 2019] Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2019.

[Nguyen *et al.*, 2018] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Proceedings of BigNet Workshop in The Web Conference (WWW)*, 2018.

[Park *et al.*, 2017] Hogun Park, John Moore, and Jennifer Neville. Deep dynamic relational classifiers: Exploiting dynamic neighborhoods in complex networks. In *Proceedings of MAISoN Workshop in International Conference on Web Search and Data Mining (WSDM)*, 2017.

[Pfeiffer *et al.*, 2015] Joseph J. Pfeiffer, III, Jennifer Neville, and Paul N. Bennett. Overcoming relational learning biases to accurately predict preferences in large scale networks. In *Proceedings of International World Wide Web Conference (WWW)*, pages 853–863, 2015.

[Rossi and Neville, 2012] Ryan Rossi and Jennifer Neville. Time-evolving relational classification and ensemble methods. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 1–13, 2012.

[Sharan and Neville, 2008] Umang Sharan and Jennifer Neville. Temporal-relational classifiers for prediction in evolving domains. In *Proceedings of International Conference on Data Mining (ICDM)*, pages 540–549, 2008.

[Xu *et al.*, 2019] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of International Conference on Learning Representations (ICLR)*, 2019.

[Yang *et al.*, 2011] Tianbao Yang, Yun Chi, Shenghuo Zhu, Yihong Gong, and Rong Jin. Detecting communities and their evolutions in dynamic social networks – a bayesian approach. *Machine learning*, 82(2):157–189, 2011.

[Zhang *et al.*, 2018] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. Timers: Error-bounded svd restart on dynamic networks. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2018.

[Zhou *et al.*, 2018] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2018.

[Zhu *et al.*, 2018] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. High-order proximity preserved embedding for dynamic networks. *IEEE Transactions on Knowledge and Data Engineering*, 30(11):2134–2144, 2018.