Inductive Representation Learning in Temporal Networks via Mining Neighborhood and Community Influences

Meng Liu
Heilongjiang University
Harbin, China
2191438@s.hlju.edu.cn

Yong Liu
Heilongjiang University
Harbin, China
liuyong123456@hlju.edu.cn

ABSTRACT

Network representation learning aims to generate an embedding for each node in a network, which facilitates downstream machine learning tasks such as node classification and link prediction. Current work mainly focuses on transductive network representation learning, i.e., generating fixed node embeddings, which is not suitable for real-world applications. Therefore, we propose a new inductive network representation learning method called MNCI by mining neighborhood and community influences in temporal networks. We propose an aggregator function that integrates neighborhood influence with community influence to generate node embeddings at any time. We conduct extensive experiments on several real-world datasets and compare MNCI with several state-of-the-art baseline methods on various tasks, including node classification and network visualization. The experimental results show that MNCI achieves better performance than baselines.

CCS CONCEPTS
- Information systems → Data mining; Social networks; Computing methodologies → Artificial intelligence; Machine learning.

KEYWORDS
Inductive Representation Learning; Temporal Networks; Neighborhood and Community Influences

ACM Reference Format:
Meng Liu and Yong Liu. 2021. Inductive Representation Learning in Temporal Networks via Mining Neighborhood and Community Influences. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR ’21), July 11–15, 2021, Virtual Event, Canada. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3404835.3463052

1 INTRODUCTION

In the real world, network data is ubiquitous such as social network, e-commerce network, and citation network, etc. By analyzing these network data, researchers can obtain user behavior to enable effective information retrieval. As a popular field, network representation learning (NRL) aims to represent a network by mapping nodes to a low-dimensional space [3]. The node embeddings generated by NRL can be used for downstream machine learning tasks such as node classification, link prediction and social search.

Related work. Based on the training goal, we can divide NRL into transductive learning and inductive learning [16]. In the early stages of NRL, most of the methods for generating node embeddings are transductive, which generate fixed node embeddings by directly optimizing the final state of the network. For example, DeepWalk [14] performs a random walk procedure over the network to learn node embeddings, node2vec [4] proposes a biased random walk procedure to balance the breadth-first and depth-first search strategy, and HTNE [21] learns node embeddings by using the Hawkes process to capture historical neighbors’ influence.

However, although transductive learning methods have good results in downstream tasks, they have difficulty adapting to the dynamic network. Many real-world tasks require node embeddings to be updated alongside network changes. Therefore, transductive learning will have to retrain the whole network to obtain new node embeddings, which is not feasible for real-world networks, especially large-scale networks.

Unlike transductive learning, inductive learning attempts to learn a model that can dynamically update node embeddings over time. For example, GraphSAGE [5] learns a function to generate embeddings by aggregating features from a node’s local neighborhood, and DyREP [16] proposes a two-time scale deep temporal point process which captures the interleaved dynamics of the observed processes for modeling node embeddings.

Our contributions. We propose a novel inductive representation learning method called MNCI to learn node embeddings in temporal networks. In temporal networks, the edges are annotated with time information, which generate fixed node embeddings by aggregating features from a node’s local neighborhood, and DyREP [16] proposes a two-time scale deep temporal point process which captures the interleaved dynamics of the observed processes for modeling node embeddings.

We believe that nodes in the network are influenced by both neighborhood and community. For neighborhood influence, it is obvious that historical neighbors of nodes will influence their future interactions. We will model the neighborhood influence from both neighbor characteristics and time information.

For community influence, we define several communities and learn an embedding for each community. Given a node, it may have different closeness to different communities. The deeper closeness the node is to a community, the more influence this community has.
According to the time information of node interaction, we can very useful for

\[ PE(u,2i) = \sin(u/10000^{2i/d}) \]
\[ PE(u,2i+1) = \cos(u/10000^{2i/d}) \]

(1)

Where \( u \) is the \( u^{th} \) node position number in \( S_{node} \), \( d \) is the
dimension size of node embedding, \( 2i \) is the \((2i)^{th}\) dimension in node
embedding, and \( PE(u,2i) \) is the encoding for the \((2i)^{th}\) dimension of the
\( u^{th} \) node embedding in \( S_{node} \). Each dimension of the positional
encoding corresponds to a sinusoid, and the wavelengths form a geometric progression from \( 2\pi \) to \( 10000 \cdot 2\pi \). Let \( PE_{u,k} \) and \( PE_u \) be
the embeddings for the \((u+k)^{th}\) node and the \( u^{th}\) node in \( S_{node} \),
respectively. For any fixed offset \( k \), \( PE_{u,k} \) can be represented as a linear function of \( PE_u \), which means that the function in Eq. (1) can
capture the relative time positions of nodes. In this way, we can
obtain the initial node embedding \( z_u^0 \) of node \( u \) at the initial time
\( t_0 \) as follows.

\[ z_u^0 = PE_u = \{PE(1,0), \ldots, PE(u,d−1)\} \]

(2)

After obtaining the initial node embedding, we can mine neigh-
borhood and community influences. Note that both influences of a
node are calculated every time it interacts with other nodes, thus
we omit the time superscript by default in the following unless we
want to distinguish two variables with different timestamps.

2.3 Neighborhood Influence

We believe that after an interaction occurs between node \( u \) and \( v \),
node \( u \) will influence the future interactions of node \( u \) with other
nodes, and \( u \) will also influence \( v \). Given a node \( u \), we assume that the
influence on \( u \) is not only related to neighbor’s own characteristics,
but also related to their interaction time. Therefore, to
mine the neighborhood influence on each node, we will analyze its
neighbors’ embedding and interaction time, respectively.

Affinity Weight. We assume that there is an affinity between
any two nodes, which reflects the closeness of their relationship.
Given a node \( u \) and its neighbor sequence \( H_u \), we can calculate \( u \)’s
affinity to different neighbors. After normalizing these affinities,
the affinity weight \( a_{u,i} \) for neighbor \( i \) on node \( u \) can be calculated as follows.

\[ a_{u,i} = \frac{\sigma(-||z_u - z_i||^2)}{\sum_{\ell \in H_u} \sigma(-||z_u - z_\ell||^2)} \]

(3)

Where \( \sigma \) is the sigmoid function, \( H_u \) is node \( u \)’s historical neigh-
bor sequence. We use negative squared Euclidean distance to measure
the affinity between two embeddings.

Temporal Embedding. In temporal networks, network structure
and node behavior will evolve over time. Thus, learning temporal
information is an important way to capture the evolutionary
process of neighborhood influence. In this part, we learn a temporal
embedding for two interactive nodes based on their interaction
timestamp. Given an interaction \((u, i, t_i)\), the temporal embedding
\( z_{t_i}^{u,i} \) between two interactive nodes at time \( t_i \) can be calculated as follows.

\[ z_{t_i}^{u,i} = F(t_c - t_i) \]

(4)
Where $t_c$ is the current time, $F(t)$ is the encoding function. For $F(t)$, we adopt random Fourier features to encode time which may approach any positive definite kernels according to the Bochner’s theorem [1, 11, 18–20].

$$F(t) = [\cos(\omega t_1), \sin(\omega t_1), \cdots, \cos(\omega d_j t_1), \sin(\omega d_j t_1)]$$

Where $\omega = \{\omega_1, \cdots, \omega_d\}$ is a set of learnable parameters to ensure that the dimension size of temporal embedding and node embedding are the same as $d$.

**Neighborhood influence embedding.** Combining affinity weight and temporal embedding, the neighborhood influence embedding $NE_u$ of $u$ at time $t_n$ can be calculated as follows.

$$NE_u = \delta_u^{NE} \sum_{i \in H_u} a_{(u,i)} z_{(u,i)}^n \odot z_i^{n-1}$$

Where $\delta_u^{NE}$ is a learnable parameter that regulates $u$’s neighborhood influence embedding, $z_i^{n-1}$ is the embedding of $u$’s neighbor $i$ at time $t_{n-1}$, $\odot$ denotes element-wise multiplication. To calculate the influence embedding of the current timestamp, we need to use the node embedding of the previous timestamp, which will be introduced later.

### 2.4 Community Influence

A community in a network is a group of nodes, within which nodes are densely connected. But nodes in different communities are sparsely linked. In real-world networks, nodes in the same community tend to have similar behavior patterns.

In this paper, we define $K$ communities $C = \{c_1, \ldots, c_K\}$ and learn an embedding for each community, where $K$ is a hyperparameter. Given a node $u$, it may have different affinity to these communities. The deeper affinity $u$ is to a community $c_k$, the more influence $c_k$ has on $u$. Let $z_u$ be the community embedding of community $c_k$. For node $u$, we calculate its affinity to all communities. Then we normalize these affinities to obtain the weights of different communities’ influence on $u$. In this case, a community $c_k$’s affinity weight $a_{(u,c_k)}$ on $u$ can be calculated as follows.

$$a_{(u,c_k)} = \frac{\sigma(-\|z_u - z_{c_k}\|^2)}{\sum_{c' \in C} \sigma(-\|z_u - z_{c'}\|^2)}$$

If a community $c_k$ has the highest affinity weight to node $u$ at time $t_n$, after updating $u$’s embedding from $z_u^{n-1}$ to $z_u^n$, we will dynamically update $c_k$’s embedding as shown in Eq. (8), i.e., we consider that $u$ belongs to $c_k$ at time $t_n$. In this way, we can obtain the community to which each node belongs at any time.

$$z_{c_k} := z_{c_k}^t - z_u^{n-1} + z_u^n$$

Finally, the community influence embedding $CO_u^n$ of node $u$ at time $t_n$ can be calculated in Eq. (9), where $\delta^{CO}$ is a learnable parameter that regulates $u$’s community influence embedding.

$$CO_u^n = \delta^{CO} \sum_{c_k \in C} a_{(u,c_k)} z_{c_k}$$

### 2.5 Aggregator Function

The GRU network [2] can capture the temporal patterns of sequential data by controlling the aggregation degree of different information and determining the proportion of historical information to be reversed. In this paper, we extend an aggregator function, which combines neighborhood and community influences with the node embeddings at the previous timestamp to generate the node embeddings at the current timestamp. The aggregator function used in this paper is defined as follows.

$$UG_u^n = \sigma(W_{UG}[z_u^{n-1} \odot NE_u^n \odot CO_u^n] + b_{UG})$$

$$NG_u^n = \sigma(W_{NG}[z_u^{n-1} \odot NE_u^n \odot CO_u^n] + b_{NG})$$

$$CG_u^n = \sigma(W_{CG}[z_u^{n-1} \odot NE_u^n \odot CO_u^n] + b_{CG})$$

$$z_u^n = \tanh(W_z[z_u^{n-1} \odot (NG_u^n \odot NE_u^n) \odot (CG_u^n \odot CO_u^n) + b_z])$$

$$z_u^n = (1 - UG_u^n) \odot z_u^{n-1} + UG_u^n \odot z_u^n$$

Where $\sigma$ is the sigmoid function, $\odot$ denotes concatenation operator, $\oplus$ denotes element-wise multiplication. $NE_u^n$, $CO_u^n$, and $z_u^n$ are neighborhood influence embedding, community influence embedding, and node $u$’s embedding at time $t_n$, respectively. $W_{UG}$, $W_{NG}$, $W_{CG}$, $W_z \in \mathbb{R}^{d \times d}$, $b_{UG}$, $b_{NG}$, $b_{CG}$, $b_z \in \mathbb{R}^d$ are learnable parameters. $U_{G_u}^n$, $NG_u^n$, $CG_u^n \in \mathbb{R}^d$ are called update gate, neighborhood reset gate, and community reset gate, respectively.

In this paper, we divide the reset gate in GRU into two reset gates, i.e., neighborhood reset gate $NG_u^n$ and community reset gate $CG_u^n$. We use $NG_u^n$ and $CG_u^n$ to control the reservation degree of neighborhood and community influence embeddings, respectively. Then, we aggregate the node embedding at the previous timestamp with reserved neighborhood and community influence embeddings to obtain a new hidden state $z_u^{n-1}$ at the current timestamp. Finally, we use $UG_u^n$ to control the reservation degree of historical information. Based on the node embedding $z_u^{n-1}$ at the previous timestamp and the new hidden state $z_u^n$ at the current timestamp, we can obtain a node embedding $z_u^n$ at the current timestamp. In this way, we can calculate node embeddings inductively.

### 2.6 Model Optimization

To learn node embeddings in a fully unsupervised setting, we apply a network-based loss function which is shown in Eq. (15) and (16), and optimize it with the Adam method [6]. The loss function encourages nearby nodes to have similar embeddings while enforcing that the embeddings of disparate nodes are highly distinct. We use negative squared Euclidean distance to measure the similarity between two embeddings. The loss function also encourages each node to have high affinity with the community it belongs to.

$$L = \sum_{u \in V} \sum_{v \in H_u} L(u, v) + \max_{u \in V} \max_{c_k \in C} \log(a_{(u,c_k)})$$

$$L(u, v) = \log \sigma(-\|z_u^n - z_v^n\|^2) - Q \cdot E_{\epsilon_0} - P_n(u) \log \sigma(-\|z_u^n - z_v^n\|^2)$$

Due to the enormous computation cost, we use negative sampling [12] to optimize the loss function as shown in Eq. (16), where $P_n(u)$ is a negative sampling distribution, $Q$ is the number of negative samples. In Eq. (15), we also introduce $\log(a_{(u,c_k)})$ to maximize the affinity between a node and the community it belongs to.
### Table 1: Node classification of all methods on all datasets

| Metric       | method    | DBLP | BITotc | BITalpha | ML1M  | AMms  | Yelp  |
|--------------|-----------|------|--------|----------|-------|-------|-------|
| **Accuracy** | DeepWalk  | 0.6140 | 0.5907 | 0.7294 | 0.6029 | 0.5780 | 0.5067 |
|              | node2vec  | 0.6249 | 0.5958 | 0.7495 | 0.6196 | 0.5772 | 0.5135 |
|              | GraphSAGE | 0.6331 | 0.6003 | 0.7389 | 0.6124 | 0.5763 | 0.5184 |
|              | HTNE      | 0.6347 | 0.5999 | 0.7635 | 0.5890 | 0.5767 | 0.5273 |
|              | DyREP     | 0.6259 | 0.6100 | 0.7430 | 0.6023 | 0.5755 | 0.5209 |
|              | MNCI      | 0.6395 | 0.6256 | 0.7842 | 0.6137 | 0.5874 | 0.5334 |
| **Weighted-F1** | DeepWalk  | 0.6107 | 0.5120 | 0.6761 | 0.5863 | 0.4252 | 0.3981 |
|              | node2vec  | 0.6210 | 0.5123 | 0.6832 | 0.5836 | 0.4248 | 0.4184 |
|              | GraphSAGE | 0.6239 | 0.5105 | 0.6750 | 0.5766 | 0.4216 | 0.4065 |
|              | HTNE      | 0.6307 | 0.5109 | 0.6806 | 0.5415 | 0.4255 | 0.4180 |
|              | DyREP     | 0.6203 | 0.5114 | 0.6843 | 0.5729 | 0.4248 | 0.4093 |
|              | MNCI      | 0.6412 | 0.5172 | 0.6859 | 0.6026 | 0.4268 | 0.4293 |

### 3 EXPERIMENTS

#### 3.1 Experimental Setup

**3.1.1 Datasets.** We conduct experiments on six real-world datasets. DBLP is a co-authorship network of computer science [21]. BITotc and BITalpha are two datasets from two bitcoin trading platforms [7, 8]. ML1M is a movie rating dataset [9]. AMms is a magazine rating dataset [13]. Yelp is a Yelp Challenge dataset [21].

**3.1.2 Baselines.** We compare MNCI with five state-of-the-art baseline methods, i.e., Deepwalk, node2vec, GraphSAGE, HTNE and DyREP. The details of these methods are described in Section 1.

**3.1.3 Parameter Settings.** We set the embedding dimension size $d$, the learning rate, the batch size, the number of negative samples $Q$, and the community number $K$ to be 128, 0.001, 128, 10, and 10 respectively, and other parameters are default values.

### 3.2 Results and Discussion

#### 3.2.1 Node Classification.** We train a Logistic Regression function as the classifier to perform 5-fold cross-validation that predicts node labels, and use Accuracy and Weighted-F1 as metrics. As shown in Table 1, MNCI outperforms all other baselines over six datasets. Compared with GraphSAGE and HTNE that use neighborhood interactions, MNCI focuses on both neighborhood and community influence, leading to further performance improvements.

#### 3.2.2 Network visualization.** We employ the t-SNE method [10] to project node embeddings on DBLP to a 2-dimensional space. In particular, we select three fields and 500 researchers in each field. Selected researchers are shown in a scatter plot, in which different fields are marked with different colors, i.e., green for data mining, purple for computer vision, blue for computer network. As shown in Figure 1, both DeepWalk, node2vec, and GraphSAGE failed to separate the three fields clearly. HTNE and DyREP can only roughly distinguish the field boundaries. MNCI separates the three fields clearly, and one of them has a clear border, which indicates that MNCI has better performance.

### 4 CONCLUSIONS

We propose an inductive network representation learning method MNCI that captures both neighborhood and community influences to generate node embeddings at any time. Extensive experiments on several real-world datasets demonstrate that MNCI significantly outperforms state-of-the-art baselines. In the future, we will study the influence of node text information on node embeddings.

### 5 ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (No. 61972135), the National Science Foundation of Heilongjiang Province in China (No. LH2020F043), and the Innovation Talents Project of Science and Technology Bureau of Harbin (No. 2017RAQXJ094).

### REFERENCES

[1] S. Bochner. 1934. A theorem on Fourier-Stieltjes integrals. *Bulletin of The American Mathematical Society* (1934), 271–277.

[2] Kyunghyun Cho, van Bart Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.
EMNLP (2014), 1724–1734.

[3] Peng Cui, Xiaow Wang, Jian Pei, and Wenwu Zhu. 2019. A Survey on Network Embedding. IEEE Transactions on Knowledge and Data Engineering (2019).

[4] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. KDD (2016), 855–864.

[5] L. William Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. NIPS (2017), 1024–1034.

[6] P. Diederik Kingma and Lei Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. international conference on learning representations (2015).

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

[8] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In KDD IEEE, 221–230.

[9] Jiacheng Li, Yujie Wang, and J. Julian McAuley. 2020. Time Interval Aware Self-Attention for Sequential Recommendation. WSDM (2020), 322–330.

[10] van der Laurens Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. Journal of Machine Learning Research (2008).

[11] Seyed Kazemi Mehran, Goel Rushab, Eghbal Sepehr, Ramanan Janahan, Sahota JaspREET, Thakur Sanjay, Wu Stella, Smyth Cathal, Poupart Pascal, and Brubaker MarcuR. 2019. Time2Vec: Learning a Vector Representation of Time. arXiv: Social and Information Networks (2019).

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. neural information processing systems (2013), 3111–3119.

[13] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Finely-Grained Aspects. EMNLP/IJCNLP (2019), 188–197.

[14] Bryan Perozzi, Rami Al-Rfou’, and Steven Skiena. 2014. DeepWalk: online learning of social representations. KDD (2014), 701–710.

[15] Hochreiter S and Schmidhuber J. 1997. Long short-term memory. Neural Computation (1997), 1735–1780.

[16] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep - Learning Representations over Dynamic Graphs. ICLR (2019).

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, N. Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. NIPS (2017), 5998–6008.

[18] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. ICLR (2021).

[19] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achana. 2019. Self-attention with Functional Time Representation Learning. NIPS (2019), 15889–15899.

[20] da Xu, chuanwei ruan, evren korpeoglu, sushant kumar, and kannan achana. 2020. Inductive representation learning on temporal graphs. ICLR (2020).

[21] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaochuan Hu, and Junjie Wu. 2018. Embedding Temporal Network via Neighborhood Formation. KDD (2018), 2857–2866.