



HAL
open science

TemporalNode2vec: Temporal Node Embedding in Temporal Networks

Mounir Haddad, Cécile Bothorel, Philippe Lenca, Dominique Bedart

► **To cite this version:**

Mounir Haddad, Cécile Bothorel, Philippe Lenca, Dominique Bedart. TemporalNode2vec: Temporal Node Embedding in Temporal Networks. COMPLEX NETWORKS 2019: 8th International Conference on Complex Networks and their Applications, Dec 2019, Lisbon, Portugal. 10.1007/978-3-030-36687-2_74 . hal-02332080

HAL Id: hal-02332080

<https://hal.science/hal-02332080>

Submitted on 24 Oct 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

TemporalNode2vec: Temporal Node Embedding in Temporal Networks

Mounir Haddad^{1,2}, Cécile Bothorel¹, Philippe Lenca¹, Dominique Bedart²

¹ IMT Atlantique, LUSI Department, UMR Lab-sticc, 29238 Brest Cedex 3, France,

² DSI Global Services, 41 avenue du Général Leclerc, 92350 Plessis-Robinson, France,

{cecile.bothorel, philippe.lenca, mounir.haddad}@imt-atlantique.fr

{dominique.bedart, mounir.haddad}@dsi-globalservices.fr

Abstract. The goal of graph embedding is to learn a representation of graphs vertices in a latent low-dimensional space in order to encode the structural information that lies in graphs. While real-world networks evolve over time, the majority of research focuses on static networks, ignoring local and global evolution patterns. A simplistic approach consists of learning nodes embeddings independently for each time step. This can cause unstable and inefficient representations over time.

We present a novel dynamic graph embedding approach that learns continuous time-aware node representations. Overall, we demonstrate that our method improves node classification tasks comparing to previous static and dynamic approaches as it achieves up to 14% gain regarding to the F1 score metric. We also prove that our model is more data-efficient than several baseline methods, as it affords to achieve good performances with a limited number of vertex representation features.

Keywords: Dynamic network embeddings, Graph representation learning, Latent representations

1 Introduction

The last decade has seen social networks flourish, as well as the data they generate. These data, usually represented using graphs, contain important information on interaction phenomena at multiple scales: it goes from the local interaction patterns between elementary entities to the global dynamics of communities.

Since several years, a multitude of research has focused on extracting networks' relevant structural information. However, in order to make graph data exploitation easier for machine learning inference models, one has to build a relevant representation of nodes/edges. Traditional machine learning approaches rely on user defined heuristics [3,22,14]. Nevertheless those task specific methods are time-consuming in terms of feature engineering. They also are without warranty for prediction tasks different than the one they were conceived for.

New research approaches known as data embeddings aim to learn data representation in low dimensional spaces [2]. These methods are data-driven as they encode data into a generic representation, independent from downstream

machine learning tasks. The early applications of data embeddings focused on text mining [16]. Regarding the advantages of such approaches, they have been adapted to graph mining among other data structures. Therefore, last years have seen a surge in graph representation learning. Some approaches are based on matrix factorization [1,4,18], on random walks [20,10,6] or on the recent advances in deep learning (auto encoders [5,23] or convolutional neural networks [11]).

Although graph embedding methods were disruptive, they avoid considering the temporal dimension. They focus on static networks where the structure of vertices and edges stays fixed among time. However, time is crucial for inference purposes in many use cases. Some approaches use temporal information for the conception of more reliable global embeddings [17,27], while other methods aim to obtain a representation for the network at each time step, and through different scales [25,15].

Literature deals with time incorporation in several ways. Embeddings can be learned independently for each time step using static methods [20,10]. Some approaches learn time steps embeddings separately and search for the optimal linear transformation of the output matrices minimizing the distance between consecutive embeddings [12]. Another approach is to learn current time step representation by initializing it with the previous embedding vectors [15]. Those different methods can be sensitive to data sparsity: for example a missing node in a time step would mislead its representation and place it far from its previous and next embeddings.

We propose TemporalNode2vec, a novel dynamic graph embedding approach, based on the static node embedding Node2vec and a smoothing mechanism for dynamic words embeddings [24]. Our main contribution lies in the way we incorporate the temporal information: the different time steps embeddings are learned jointly. We show in our experiments that this method improves node classification tasks comparing to relevant baseline algorithms. We also interpret the hyper parameters influence, especially the embedding dimension.

The remainder of this paper is organized as follows. In Section 2, we provide the problem definition. Section 3 describes how we set up TemporalNode2vec. In section 4, we expose the performed experiments and interpret the results.

2 Problem definition

We consider a network composed of $|V|$ vertices $V = \{v_1, \dots, v_{|V|}\}$ connected by timestamped interactions $I = \{i_{j,k,t}\}$ where $i_{j,k,t}$ represents an interaction between v_j and v_k at t . The goal of temporal embedding is to have a representative vector of each vertex at each time step. However, regarding the number of the different timestamps (sometimes equal to the number of interactions), it is obvious that there should be some aggregation over time: we consider T time slices (referred by time steps in the rest of the paper) rather than instant moments. Within each time step, we aggregate the interactions between pairs of vertices, to form weighted edges (weights represent the number of grouped interactions). Formally, the whole input dataset is assumed to be a set of T

graphs $\{G_1, \dots, G_T\}$ where G_t is the undirected weighted graph representing the interactions of the t -th time step. For each time step t ($t \in \llbracket 1, T \rrbracket$), we aim to learn mapping functions $f_t : V \rightarrow \mathbb{R}^d$. Here, d is the embedding dimension ($d \ll |V|$). The output expected is a set of T matrices of size $d \times |V|$ representing the learned mappings, i.e. the t -th matrix (let U_t be this matrix in the rest of the paper) contains the embedding vectors of all vertices at time step t .

3 TemporalNode2vec

3.1 Random walks

The main idea behind the very first graph embedding approaches is adapting word embedding techniques that offer remarkable improvement in terms of inference comparing to previous work. For that purpose, one has to transform graph data to *sentences* of vertices. DeepWalk comes with a method to build those sequences of vertices called *walks*, inspired by early applications of random walks [19]. The idea is to simulate a probabilistic *walk* all over a graph vertices through its (un)directed (un)weighted edges. Node2vec comes with an extension of random walks by offering more flexibility to the way nodes neighborhoods are explored, i.e. whether to privilege local exploration. In our method, we build our random walks in the same fashion. This means that, given a sequence of temporal graphs $\{G_1, \dots, G_T\}$, we compute T sets of random walks. We then obtain $T \times N \times |V|$ sequences of vertices of length l , where N is the number of walks starting from each vertex.

3.2 PPMI matrices

In the literature of words embeddings, we observe that similar words have similar neighboring words [7]. This property, called homophily, is also present in social graphs [8]. One way to capture nodes neighboring structures uses statistics on the frequencies by which vertices co-appear in the set of random walks considered [5]. We define the Positive Pointwise Mutual Information matrix (PPMI) as follows:

$$PPMI_t(v_1, v_2) = \max \left(0, \log \left(\theta \frac{|v_1, v_2|_t^w \cdot |V|}{|v_1|_t \cdot |v_2|_t} \right) \right) \quad (1)$$

$$\forall (v_1, v_2 \neq v_1, t) \in V^2 \times \llbracket 1, T \rrbracket$$

where $|v_1, v_2|_t^w$ is the number of times v_1 and v_2 co-appear in the set of walks of G_t within a window of size w , $|v_i|_t$ is the number of occurrences of v_i in the set of walks of G_t , and θ is a tunable hyper parameter. The *PPMI* matrices can be seen as temporal similarity matrices. It is worthwhile to mention that we use *positive PMI* matrices in order to reduce instability, as $\log \left(\theta \frac{|v_1, v_2|_t^w \times |V|}{|v_1|_t \times |v_2|_t} \right)$ can produce large negative values, which illustrate rare co-appearance, assimilable to zero co-appearance. The trade-off between stability and rare co-appearance is controlled by θ . Thresholding *PMI* matrices in this way is also used in [13].

3.3 Objective function components

Several static graph embedding techniques aim to learn U_t matrices satisfying:

$$u_1(t)^T u_2(t) \approx PPMI_t(v_1, v_2) \quad (2)$$

where $u_i(t)$ is the embedded vector of v_i at time step t . Node2vec uses negative sampling to implicitly satisfy (2). [13] observes that it is equivalent to low-rank factorization of PMI matrices, shifted by a constant (θ in equation (1)). Therefore, the static term L_{St} of our objective function can be represented as:

$$L_{St} = \sum_{t=1}^T \|PPMI_t - U_t U_t^T\|_F^2 \quad (3)$$

An important issue of learning temporal embeddings of a dynamic network is called alignment: the learned temporal vectors, encoding nodes/edges, may not be placed in the same latent space over time. To conceive a dynamic embedding method, one has to deal with alignment, since L_{St} is invariant to U_t matrices different rotations. Thus, it is necessary to add a smoothing term L_{Sm} to our objective function:

$$L_{Sm} = \sum_{t=2}^T \|U_t - U_{t-1}\|_F^2 \quad (4)$$

After adding a final term L_{LR} standing for low-rank data-fidelity enforcement as adopted in [24], the overall objective function to minimize is as follows:

$$\begin{aligned} L &= L_{St} + \tau L_{Sm} + \lambda L_{LR} \\ &= \sum_{t=1}^T \|PPMI_t - U_t U_t^T\|_F^2 + \tau \sum_{t=2}^T \|U_t - U_{t-1}\|_F^2 + \lambda \sum_{t=1}^T \|U_t\|_F^2 \end{aligned} \quad (5)$$

3.4 Objective function optimization

As common temporal networks contain tens of thousands of vertices, $PPMI$ matrices may not fit in memory. Minimizing the objective function may require additional simplifications. In the same fashion as [24], we introduce W_t matrices and γ hyper parameter to break the symmetry of $PPMI$ factorization:

$$\begin{aligned} &\sum_{t=1}^T \|PPMI_t - U_t W_t^T\|_F^2 + \lambda \sum_{t=1}^T \|U_t\|_F^2 + \lambda \sum_{t=1}^T \|W_t\|_F^2 \\ &+ \tau \sum_{t=2}^T \|U_t - U_{t-1}\|_F^2 + \tau \sum_{t=2}^T \|W_t - W_{t-1}\|_F^2 + \gamma \sum_{t=1}^T \|U_t - W_t\|_F^2 \end{aligned} \quad (6)$$

Minimizing (6) for U_t (equivalently for W_t) is solvable by simple 0 gradient, i.e. $U_t M = N$, where

$$\begin{aligned} M &= W_t^T W_t + (\lambda + 2\tau + \gamma) I \\ N &= PPMI_t W_t + \gamma W_t + \tau (U_{t-1} + U_{t+1}) \end{aligned}$$

for $t \in \llbracket 2, T-1 \rrbracket$ and constants adjusted for $t \in \{0, T\}$. We use block coordinate descent in order to solve $U_t M = N$ to afford scaling large $PPMI$ matrices, as only blocks of U_t (equivalently W_t) are updated at each time, therefore only blocks of $PPMI_t$ are loaded in memory at each time.

3.5 Temporal embeddings initialization

As $PPMI$ are a sort of temporal similarity matrices, they can be used during the initialization step in order to speed up the convergence of our method. Thus, for $t \in \llbracket 1, T \rrbracket$, we initialize U_t and W_t by choosing carefully d rows of $PPMI_t$. We choose the top d l1-normalized rows maximizing the variance: a high variance means a better discrimination between nodes, and the l1-norm stands for normalizing regarding nodes appearances.

Algorithm 1: TemporalNode2vec

```

Data:  $(\{G_1, \dots, G_T\}, V, d, l, w, p, q, \theta, \lambda, \gamma, \tau, iter)$ 
for  $t \leftarrow 1$  to  $T$  do
     $walks = generateWalks(G_t, l, p, q);$ 
     $PPMI_t = computePPMI(walks, w, \theta);$ 
end
 $\{U_1, \dots, U_T, W_1, \dots, W_T\} = initialize(PPMI, d);$ 
for  $i \leftarrow 1$  to  $iter$  do
    for  $t \leftarrow \llbracket 1, T \rrbracket$  do
        for  $b \leftarrow batch(V)$  do
             $U_t[b] = update(W_t[b], \lambda, \gamma, \tau, PPMI_t);$ 
             $W_t[b] = update(U_t[b], \lambda, \gamma, \tau, PPMI_t);$ 
        end
    end
end
return  $\{U_1, \dots, U_T\}$ 

```

4 Experiments

4.1 Baseline methods

As seen above, our algorithm has multiple hyper parameters: $\{l, w, p, q, \theta, \lambda, \gamma, \tau\}$. Testing 3 values for each over a grid search would lead to more than 6k sets of parameters. To workaround this issue, we choose to look for optimal parameters one by one, by fixing the others. Table 1 summarizes the tested values. To evaluate TemporalNode2vec, we consider 3 state-of-the art baseline methods.

- **DeepWalk** [20]: This method learns representations of a static graph nodes using random walks over edges. It has two hyper parameters: the walk length l and the window size w . For our experiments, we test a grid search over $(l, w) \in \{20, 40, 60, 80\} \times \{4, 8, 12, 16\}$.

- **Node2vec** [10]: Node2vec extends DeepWalk’s way of discovering nodes neighborhood by adding two hyper parameters (p, q) : the return and the in-out parameters. We keep the values of l and w giving the best performances in DeepWalk tests, and perform a grid search over $(p, q) \in \{0.5, 1, 1.5, 2, 5\}^2$.
- **DynamicTriad** [25]: DynamicTriad is a dynamic representation learning approach focusing on how triads of vertices close, i.e how a pair of nodes sharing a common neighbor vertex can connect over an edge. This method has two hyper parameters: β_0 which is the weight of triad closure process and β_1 the temporal smoothing parameter. For our benchmarking experiments, we refer to DynamicTriad paper tests and consider a grid search over $(\beta_0, \beta_1) \in \{0.01, 0.1, 1, 10\}^2$.

Also, there are few other dynamic approaches we do not consider in our comparison due to the unavailability of the code (Dynnode2vec [15]) or their proven inefficiency³ (Temporal network embedding [26]).

Concerning the embedding dimension, we have chosen $d = 48$ for the sake of memory space. Smaller values of d are tested in section 4.7.

4.2 Datasets

To compare the different algorithms embeddings performances, we gathered 3 real-world datasets. These datasets must meet some criteria. They should consist of temporal networks, i.e. sets of timestamped graphs. Also, the datasets must include metadata showing the existing evolving ground-truth communities (called labels in the rest of the paper). Part of the evaluation determines whether embedding approaches preserve community membership as much as possible.

- **AMiner**: This dataset⁴ consists of 51k researchers and 624k coauthor relationships. It is divided into 17 timestamped weighted graphs, where weights represent the number of common co-authorships between a pair of researchers within a time step. As articles are published in conferences addressing different research fields, it is possible to assign labels to authors: at each time step, we assume that a researcher belongs to a community (i.e. research field) if the majority of his papers are published in related conferences. Also, for the experiments made for embedding dimension analysis, we consider 3 samples of different sizes of this dataset (Table 2).
- **Yelp**: This dataset is an extract of Yelp⁵ challenge dataset. It traces internet users comments on businesses (like restaurants and malls). We consider users and businesses as being nodes, and comments are regarded as interactions. Businesses are assigned with categories. We keep only the top eight categories for our experiments. As businesses categories do not change over time, we assign labels for users: within each time step, a user is assumed to belong to a category if the majority of his comments are made on this category related

³ [25] proved its inefficiency comparing to later graph embedding methods

⁴ We use DynamicTriad [25] derived version of ArnetMiner [21].

⁵ <https://www.yelp.com/dataset/challenge>

businesses. Finally, our whole temporal graph is divided into 17 timestamped graphs, with a total of 744k edges between 38k nodes.

- **Tmall** : This dataset is an extract of the sales at Tmall.com⁶ 6 months before the "Double 11 Day" event in 2014. It stores buyers interactions with products (i.e. click on product page, add to cart, purchase or add to favourites). Products are assigned with categories. For our experiments, we consider only the top five categories. We assign labels to users as described for Yelp dataset. We obtain a temporal graph divided into 10 timestamped graphs, with a total of 2.9M edges between 27k nodes.

Parameter	Tested Values
l	{20, 40, 60, 80}
w	{4, 7, 10, 13, 16}
p	{0.2, 0.5, 1, 2, 5}
q	{0.2, 0.5, 1, 2, 5}
θ	{0.05, 0.25, 4, 20}
λ	{0, 0.1, 0.3, 1, 3, 10}
γ	{0.1, 0.3, 1, 3, 10}
τ	{0, 0.1, 0.3, 1, 3, 10}

Table 1: Model tested values

Dataset	nodes	nodes	time steps
AMiner	51 060	624 381	
<i>Smp_A</i>	16 546	12 826	
<i>Smp_B</i>	9 244	5 884	17
<i>Smp_C</i>	2 300	1 230	
Yelp	38 475	744 195	17
Tmall	27 039	2 976 392	10

Table 2: Datasets

4.3 Application scenarios

We challenge our approach and baseline methods by executing different inference tasks on the basis of their respective embeddings, regarding to the F1 score metric. As those tasks are identical for the compared methods, better scores reflect more efficient embeddings. Following inference tasks have been experimented:

- Node classification: A classifier is trained on nodes embeddings to find their labels. Logistic regression is used as the classification method.
- Node class prediction: This task is similar to node classification. We predict nodes labels at a time step using their embeddings at the previous time step.
- Edge reconstruction: In this task, the goal is to determine whether there is an edge between a pair of nodes, based on the distance between their embeddings.
- Edge prediction: This task is similar to edge reconstruction with the difference that we aim to predict the presence of edges between pairs of nodes regarding their previous embeddings.

4.4 Comparison results

Table 3 shows the results of the experiments made using the different considered methods. It demonstrates that TemporalNode2vec outperforms other baseline

⁶ <https://tianchi.aliyun.com/competition/entrance/231576/information>

methods in the tasks related to node classification (up to 14.2% in terms of F1 score). This means that learning the representations jointly across all time steps improves the overall performances, as we force embeddings continuity over time.

On the other hand, our approach is less efficient in edge related inference tasks⁷. We interpret those results as following: TemporalNode2vec captures better the global temporal structures than the local ones, as edge reconstruction/prediction tasks focus on pairs of nodes independently from the rest of the whole network, while node classification tasks consider the overall placement of the embeddings. Thus, our model seems to be more suitable for community detection and capturing their evolution. Concerning the efficiency of DynamicTriad at the edge prediction task, we suppose that it is due to the algorithm core idea, as DynamicTriad focuses on edges dynamics, i.e. how triads close/open. Finally, further investigation is needed to explain Node2vec performances on the edge reconstruction task.

Dataset	Algorithm	Node classification	Node class prediction	Edge reconstruction	Edge prediction
AMiner	DeepWalk	0.69884	0.67333	0.92628	0.75845
	Node2vec	0.77080	0.75846	0.98362	0.80619
	DynamicTriad	0.74432	0.73967	0.92363	0.87294
	TemporalNode2vec	0.87999	0.85189	0.86086	0.74929
Yelp	DeepWalk	0.29759	0.25357	0.94453	0.75710
	Node2vec	0.29747	0.26062	0.98831	0.79555
	DynamicTriad	0.25554	0.24698	0.93704	0.91593
	TemporalNode2vec	0.31512	0.28118	0.93214	0.81067
Tmall	DeepWalk	0.94436	0.68835	0.84765	0.68754
	Node2vec	0.96451	0.70008	0.91305	0.72787
	DynamicTriad	0.56336	0.53252	0.77821	0.73260
	TemporalNode2vec	0.96632	0.77306	0.77289	0.68043

Table 3: Models scores

4.5 Hyper parameter analysis

We also analyze the impact of each hyper parameter on the overall method performance. From figure 1 we can see that the most sensitive parameters are q (walk return parameter) and θ (rare co-appearance / stability trade-off parameter).

Furthermore, there are two interesting observations we can notice. First, larger widow sizes give better results. This may confirm the idea that our model is more suitable for finding global temporal structures of networks, and consequently giving better results on node classification/class prediction tasks; as a matter of fact, while considering large values of w , vertices neighborhoods are not confined to their immediate neighbors. The second observation concerns the walk length l . We notice that smaller walks give better results. This may be due to the fact that small l values output more balanced walks over nodes. For example, if we imagine dense regions (in terms of edges and weights) in our temporal graph, a long random walk may be unable to step out of it.

⁷ More tuning of the hyper parameters (especially p and q) may improve edge reconstruction / prediction tasks results.

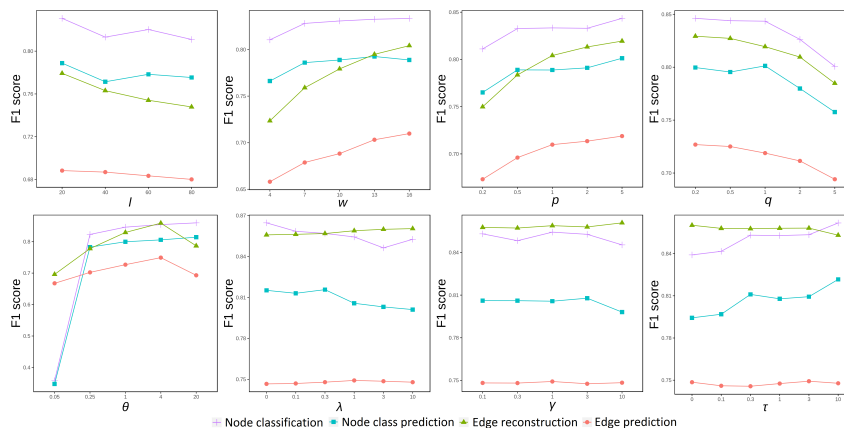


Fig. 1: Hyper parameter analysis

4.6 Alignment

Alignment is an important aspect in dynamic networks embedding, e.g. for visualization purposes where the position of nodes has to be stable in the coordinate referential. In order to challenge the different baseline methods alignment, embeddings are submitted to a test consisting on predicting nodes class change, based on the difference of consecutive embeddings: the ground-truth communities of the considered datasets have a semantic meaning that we reasonably assume to be constant over time (research fields, categories of products and businesses). Consequently, if alignment is respected, communities locations in the embedding latent space should be stable as well, or continuous at least. Thus, for good aligned methods, a node’s trajectory in the embedding space gives information about its future class. Then, given the difference vector of a node’s consecutive embeddings (input data), it should be possible to predict the class change of a node (target label). Table 4 shows the results of this experiment. We observe that the dynamic embedding methods outperform the static ones, proving alignment importance. However, the static methods present better results than expected, suggesting that they might include some inherent *normalization* mechanism of the output matrix, forcing alignment. Further investigation is needed to explain this point.

Algorithm	AMiner	Yelp	Tmall
DeepWalk	0.48921	0.40369	0.40710
Node2vec	0.48264	0.41124	0.43139
DynamicTriad	0.52823	0.48088	0.49889
TemporalNode2vec	0.54446	0.46089	0.56838

Table 4: Node class change prediction f1-score

4.7 Embedding dimension

The target latent space dimension is an important hyper parameter: the strength of embedding algorithms lies in inference performance, but also on how efficient their data encoding process is. For that purpose, we compare TemporalNode2vec to the considered baseline methods regarding the influence of the embedding dimension for the node classification task. Figure 2 shows the results of our experiment: while TemporalNode2vec affords to have good performances with a very limited number of features (from $d = 10$), the other approaches need a consequent embedding dimension and are unstable when dimension grows.

Furthermore, we notice a sort of saturation regarding the embedding dimension: it seems that 20 features are sufficient to describe nodes neighborhoods. This value may be the *intrinsic latent dimensionality* of our temporal graph, i.e. the minimal latent space able to capture all structural information present in a graph. It is related to the nature of the input data and do not depend on its size [9]. To confirm this idea, we perform more experiments. Figure 3 shows the impact of the embedding dimension on the inference score (node classification) on the three samples of different sizes previously described in table 2. We observe that the shapes of the curves are similar. In other words, the ratio between the information encoded at a dimension d and the *maximum* information we can capture do not depend on the sample size.

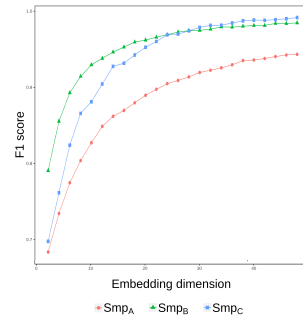
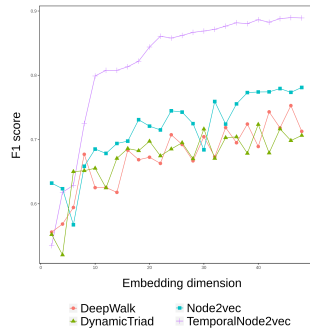


Fig. 2: Models data efficiency Fig. 3: Data efficiency on samples

5 Conclusion

In this paper, we presented TemporalNode2vec, a novel approach of dynamic networks embedding. We compared our algorithm to several state-of-the-art techniques and showed its efficiency in node classification tasks. Furthermore, we demonstrated the data encoding efficiency of our approach, as we showed that, in a real-world dataset consisting of more than 51k nodes, a representation of 20 features is sufficient for gathering relevant temporal and structural characteristics of the network. Further work tied up to embeddings exploitation for different purposes (dynamics analysis and prediction, visualization) is under way.

References

1. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: *Advances in neural information processing systems*. pp. 585–591 (2002)
2. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35(8), 1798–1828 (2013)
3. Bhagat, S., Cormode, G., Muthukrishnan, S.: Node classification in social networks. In: *Social network data analytics*, pp. 115–148. Springer (2011)
4. Cao, S., Lu, W., Xu, Q.: Grarep: Learning graph representations with global structural information. In: *Proceedings of the 24th ACM international on conference on information and knowledge management*. pp. 891–900. ACM (2015)
5. Cao, S., Lu, W., Xu, Q.: Deep neural networks for learning graph representations. In: *Thirtieth AAAI Conference on Artificial Intelligence* (2016)
6. Chen, H., Perozzi, B., Hu, Y., Skiena, S.: Harp: Hierarchical representation learning for networks. In: *Thirty-Second AAAI Conference on Artificial Intelligence* (2018)
7. Firth, J.R.: A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis* (1957)
8. Fortunato, S.: Community detection in graphs. *Physics reports* 486(3-5), 75–174 (2010)
9. Granata, D., Carnevale, V.: Accurate estimation of the intrinsic dimension using graph distances: Unraveling the geometric complexity of datasets. *Scientific reports* 6, 31377 (2016)
10. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. pp. 855–864. ACM (2016)
11. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016)
12. Kulkarni, V., Al-Rfou, R., Perozzi, B., Skiena, S.: Statistically significant detection of linguistic change. In: *Proceedings of the 24th International Conference on World Wide Web*. pp. 625–635. *International World Wide Web Conferences Steering Committee* (2015)
13. Levy, O., Goldberg, Y.: Neural word embedding as implicit matrix factorization. In: *Advances in neural information processing systems*. pp. 2177–2185 (2014)
14. Liben-Nowell, D., Kleinberg, J.: The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58(7), 1019–1031 (2007)
15. Mahdavi, S., Khoshraftar, S., An, A.: dynnode2vec: Scalable dynamic network embedding. In: *2018 IEEE International Conference on Big Data (Big Data)*. pp. 3762–3765. IEEE (2018)
16. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013)
17. Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Continuous-time dynamic network embeddings. In: *Companion of the The Web Conference 2018 on The Web Conference 2018*. pp. 969–976. *International World Wide Web Conferences Steering Committee* (2018)
18. Ou, M., Cui, P., Pei, J., Zhang, Z., Zhu, W.: Asymmetric transitivity preserving graph embedding. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. pp. 1105–1114. ACM (2016)

19. Pearson, K.: The problem of the random walk. *Nature* 72(1867), 342 (1905)
20. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710. ACM (2014)
21. Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., Su, Z.: Arnetminer: extraction and mining of academic social networks. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 990–998. ACM (2008)
22. Vishwanathan, S.V.N., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. *Journal of Machine Learning Research* 11(Apr), 1201–1242 (2010)
23. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1225–1234. ACM (2016)
24. Yao, Z., Sun, Y., Ding, W., Rao, N., Xiong, H.: Dynamic word embeddings for evolving semantic discovery. In: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. pp. 673–681. ACM (2018)
25. Zhou, L., Yang, Y., Ren, X., Wu, F., Zhuang, Y.: Dynamic network embedding by modeling triadic closure process. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
26. Zhu, L., Guo, D., Yin, J., Ver Steeg, G., Galstyan, A.: Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering* 28(10), 2765–2777 (2016)
27. Zuo, Y., Liu, G., Lin, H., Guo, J., Hu, X., Wu, J.: Embedding temporal network via neighborhood formation. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 2857–2866. ACM (2018)