On Structural vs. Proximity-based Temporal Node Embeddings

Puja Trivedi*, Alican Büyükçakır*, Yin Lin, Yinlong Qian, Di Jin, Danai Koutra
{pujat,alicanb,irenelin,yinlongq,dijin,dkoutra}@umich.edu
University of Michigan

ABSTRACT

We investigate the representation power of static node embeddings in dynamic or temporal settings. To this end, we introduce a framework that incorporates different design options for extending static node embeddings to temporal settings: temporal combination schemes to introduce dynamics in otherwise static approaches, alignment methods that lead to comparability of embedding dimensions across time steps, and different edge operators for generating edge embeddings from node embeddings. In our empirical analysis, we evaluate the performance of both proximity-based and structural node embedding methods in a temporal link prediction task over four time-evolving networks. Our results show that proper choice over these designs yields up to 20% absolute improvement over baselines that do not leverage temporal combination and embedding alignment. We further present broad trends to guide design decisions for embedding methods in temporal settings.

KEYWORDS

temporal graphs, graph embeddings, temporal link prediction

ACM Reference Format:
Puja Trivedi*, Alican Büyükçakır*, Yin Lin, Yinlong Qian, Di Jin, Danai Koutra. 2018. On Structural vs. Proximity-based Temporal Node Embeddings. In *MLG '20: 16th International Workshop on Mining and Learning with Graphs, Aug 24, 2020, San Diego, CA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Representation learning on static graphs is a well-studied problem [8, 20, 30, 31] and is central to many graph mining applications, including community detection [18], recommendation systems [2], and information retrieval [13]. However, many networks naturally evolve over time: for example, interactions on social media such as Facebook or Snapchat continually evolve, email and other communication networks grow continuously by amassing more and more email exchanges. Therefore, there has been a recent upsurge of interest in dynamic representation learning on graphs for various tasks, including temporal link prediction [19], user-state change prediction [16] and identity stitching [14]. However, it is worth noting that the promising performance of these customized models comes with the trade-off of complexity by introducing the latent features that capture temporal dependency over time [7], or graph-level attention [24] across graph time-streaming. Static methods, on the other hand, typically are more computationally-efficient.

Motivated by this trade-off and the computational efficiency of static methods, we seek to explore the potential of static methods in dynamic scenarios. To this end, we introduce a framework that considers different design options, including techniques to combine embeddings over time, align embeddings across time-stamps, and combine node embeddings via edge operators, to induce temporal information. In our analysis, we consider embedding methods that fall into two categories: proximity-based and structural embeddings, where the former generate embeddings that preserve closeness/communities, and the latter generate embeddings that preserve structural similarity or roles [23]. Thus, we also investigate how proximity-vs-structural paradigms influence representation power in this setting.

Adapting embeddings from a static setting to a dynamic setting comes with several challenges. First, embeddings are generated at each snapshot separately, and it is unclear how to extract temporal dynamics from these distinct embeddings. Therefore, we consider three different temporal combination schemes that incorporate the information gains from preceding time steps (including combining the static embeddings from different snapshots with exponential decay, putting more emphasis on the recent embeddings). Second, before the embeddings can be combined across time, their manifolds must be comparable. While structural embeddings which capture role information are comparable across graphs and time stamps, proximity-based (community-based) embeddings which capture homophily and closeness are not comparable [10, 23]. To address this challenge, we draw inspiration from recent work in NLP [9] and temporal graph embedding alignment [28] that uses the solution to the Orthogonal Procrustes Problem [25] to align the embeddings across different time stamps. Additionally, following [29], we also take the average over proximity-based embeddings to represent structural properties, which are comparable across graphs. Third, after generating temporally-combined, aligned node-embeddings, we consider three different operators to induce edge embeddings (including concatenating, and averaging the endpoints' node embeddings). We evaluate the cross product of these three designs in our general framework on a temporal link prediction task. Our contributions are summarized as follows:

- **Framework.** We introduce a framework that incorporates a variety of sensible design options for converting static node embeddings into temporal embeddings that can be used in downstream temporal tasks.
- **Empirical evaluation.** We analyze four static proximity-based and structural node embeddings approaches over four datasets in a temporal link prediction task, and discuss the representation power of the different design choices.

*The authors contributed equally.

MLG '20, Aug 24, 2020, San Diego, CA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

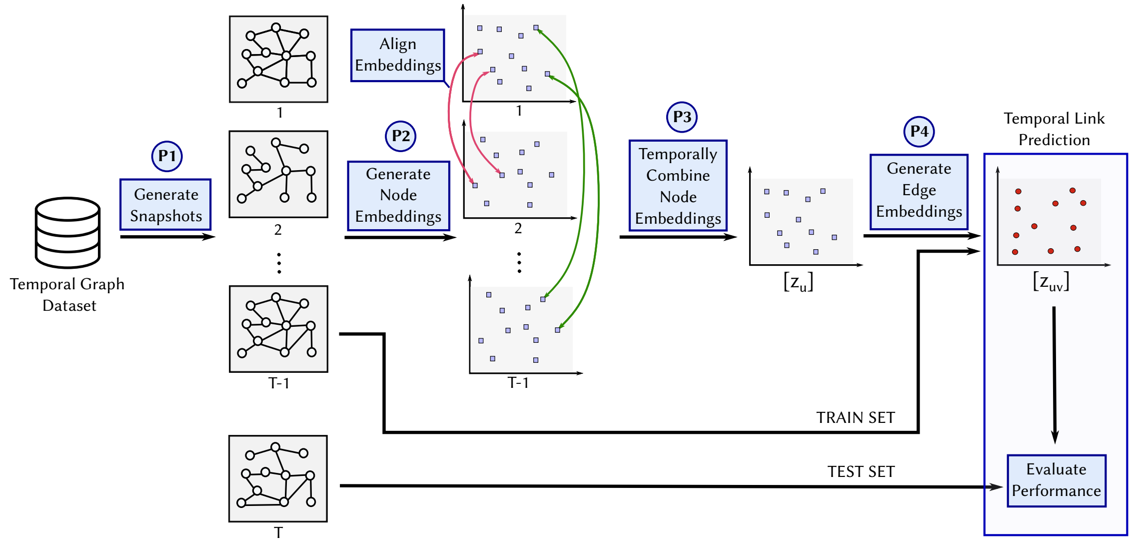

Figure 1: Proposed pipeline for exploring the impact of structural and proximity-based node embeddings in temporal graphs. A temporal graph is first pre-processed and split into different temporal granularities. Then, different embedding methods are applied at each graph snapshot, and these embeddings are aligned and fused into one. The fused embeddings are used for generating edge embeddings which can be used to train classifiers on the task of temporal link prediction.

2 RELATED WORK

Our work is related to node representation learning in static and dynamic graphs, which we discuss next.

Node embeddings in static graphs. Existing work on learning representations on static graphs (i.e. graph embedding methods) can be examined in two main groups [23]. The first group is **proximity-based embeddings** (e.g. DeepWalk [20], LINE [30], node2vec [8] and NetMF [21]) where closeness and homophily among the nodes are captured; and the second group is **structural embeddings** (e.g. GraphWave [5], node2bits [14], rolX [12] and xNetMF [11]) where roles and structural similarity of the nodes are captured. For a detailed discussion on the structural and proximity-based embeddings, the reader is referred to [23]. In our work, we pick two structural (GraphWave, xNetMF) and two proximity-based methods (LINE, node2vec) to generate embeddings at each graph snapshot.

Node embeddings in dynamic graphs. Learning representations of dynamic and temporal graphs is a research area that has recently gained a lot of interest [32]. Previous methods consider various approaches including: temporal random walks (CTDNE [19], node2bits [14]), Hawkes Processes (HTNE [33]), LSTMs with autoencoders (dyngraph2vec [6]), and predicting trajectories of individual node embeddings (JODIE [16]) to incorporate temporal dynamics into the learned embeddings. However, these methods use mechanisms for representing dynamics that are inherent to their method construction. In other words, it is not possible to benefit from previously computed and well-performing static node embeddings using these methods. Furthermore, dynamic methods may require computationally expensive techniques to encode temporal properties, such as LSTMs. tNodeEmbed [28] utilizes node2vec to initialize its embeddings (though any static method can be used), aligns consecutive timestamps, and sequentially optimizes two objective functions jointly, one for preserving neighborhood between nodes and another for capturing temporal aspect. Unlike this work, tNodeEmbed uses expensive LSTMs to obtain the temporal combination of node embeddings, and uses only concatenation to generate edge embeddings from the node embeddings.

Table 1: Summary of notation

Symbol	Definition
$\mathcal{G} = \{G_1, G_2, \dots\}$	a graph time-series
$G_t = (\mathcal{V}_t, \mathcal{E}_t)$	a directed and weighted temporal network from $\mathcal{G}$ with $\lvert\mathcal{V}_t\rvert$ nodes and $\lvert\mathcal{E}_t\rvert$ temporal edges
$\mathbf{A}_t$	adjacency matrix for graph G_t at time t
α	the decay factor in the temporal summary graph model
θ	the decay factor in the temporal embedding smoothing
d	dimensionality of the embedding
$\mathbf{Z}$	$\lvert\mathcal{V}\rvert \times D$ embedding matrix

3 NOTATION

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau)$ be a temporal network where $\mathcal{V}$ is the set of vertices, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \mathbb{R}^+$ is the set of temporal edges between vertices $\mathcal{V}$, and $\tau : \mathcal{E} \to \mathbb{R}^+$ is a function mapping edges to timestamps. In temporal settings, timestamped edges are processed in batches containing sets of edges that arrive at equidistant time intervals. The length of these time intervals is referred as the *granularity* of the temporal data. Let T be the total number of batches with the choice of a granularity (e.g. days, weeks, months). Then, each i $(1 \le i \le T)$ generates a *graph snapshot*, G_i, where edges of G_i are the edges that arrive only in the batch i. That is, $G_i = (\mathcal{V}, \mathcal{E}_i)$, where $\mathcal{E}_i = \{e \mid e \in \textit{batch } i\}$.

Embedding $\mathbf{Z} : \mathcal{V} \to \mathbb{R}^d$ maps each vertex of $\mathcal{G}$ to d-dimensional feature representations where $d << |\mathcal{V}|$. At each graph snapshot G_i, a corresponding embedding $\mathbf{Z}_i$ may be generated. The preceding embeddings may be partially retained to contribute to the current embedding or be completely discarded.

4 PROPOSED FRAMEWORK

In this work, we consider the problem of dynamic graph representation learning by the fusion of static embeddings over time. Specifically, we focus on different *types* of static embeddings, factors that affect their performance, and their behavior under the incorporation of different temporal dynamics.

To explore the impact of discrete techniques on dynamic network representation learning, we divide our framework into four phases: (P1) generating snapshots, (P2) generating node embeddings, (P3) temporal combination and (P4) generating edge embeddings. We give an overview of our framework in Figure 1.

4.1 (P1) Generating Snapshots

Given timestamped edges, network snapshots can be generated by defining discrete time periods that are sensible for the domain or application at hand. Typical time periods in the literature include hourly, daily, weekly, monthly, or yearly granularities [1, 15, 26, 27].

4.2 (P2) Generating Node Embeddings

For a given embedding method, we generate node embeddings at every snapshot G_i. Since proximity-based embeddings are not comparable across time-stamps or graphs [3, 10], we consider alignment techniques that aim to align the embedding spaces that are learned independently per snapshot.

- **Averaging Embeddings over Multiple Runs**: Srivinasan and Ribeiro [29] showed that averaging the proximity representations over multiple runs leads to structural embeddings, and hence, a form of an alignment. Following that, we average over 3 runs to generate aligned embeddings.

$$\hat{\mathbf{Z}}_i = \frac{1}{3} * (\mathbf{Z}_i^1 + \mathbf{Z}_i^2 + \mathbf{Z}_i^3)$$

- **Procrustes**: Inspired by [3, 9], we use Procrustes to find the transformation matrix, $\mathbf{R}_i \in \mathbb{R}^{d \times d}$, per timestep i that minimizes:

$$\mathbf{R}_i = \min_{\mathbf{Q}^T\mathbf{Q}=I} \|\mathbf{Z}_i\mathbf{Q} - \mathbf{Z}_k\|_2^2, \text{ for } i = 1, \ldots, T-1$$

  Then, $\hat{\mathbf{Z}}_i = \mathbf{Z}_i\mathbf{R}_i$ is rotationally aligned to the embeddings $\mathbf{Z}_k$ (at timestep k), and the embeddings are more comparable across time-steps.

4.3 (P3) Temporal Combination

Each embedding is generated statically at each snapshot. Therefore, while the embedding captures graph properties, it does not represent how the graph changed over time. By fusing or combining the time series of embeddings, we induce temporal dynamics. After alignment, we consider the following temporal-combination techniques:

- **No Combination**: Use $\mathbf{Z}_{NONE} = \mathbf{Z}_T$, i.e. use only the last embedding that is generated, and disregard the preceding ones.
- **Linear Combination**: $\mathbf{Z}_{LIN} = \sum_i^T \alpha \mathbf{Z}_i$, $\alpha \in \mathbb{R}$. This is equivalent to summing up embeddings when $\alpha = 1$, which is what we use in our experiments.
- **Exponential Decay**: $\mathbf{Z}_{EXP} = \sum_i^T e^{-\theta(T-i)}\mathbf{Z}_i$, i.e. embeddings of past snapshots are weighted exponentially less than more recent ones. In our experiments, we use $\theta = 0.3$.

4.4 (P4) Generating Edge Embeddings

Having the node embeddings $\mathbf{z}_u$ and $\mathbf{z}_v$ for the nodes u and v, we consider 3 combination methods that transform $\mathbf{z}_u$ and $\mathbf{z}_v$ into an edge embedding $\mathbf{z}_{uv}$.

- **Concatenation**: Use $\mathbf{z}_{uv} = (\mathbf{z}_u || \mathbf{z}_v)$, i.e. concatenate the two node embeddings to generate the edge embedding. Previously, this was adopted in [28].
- **Average**: $\mathbf{z}_{uv} = (\mathbf{z}_u + \mathbf{z}_v)/2$, i.e. take the average of the two node embeddings.
- **Hadamard Product**: $\mathbf{z}_{uv} = (\mathbf{z}_u \odot \mathbf{z}_v)$, i.e. take the piece-wise multiplication of the entries of the two node embeddings. Previously, this was recommended in [8] and adopted in [24].

5 EMPIRICAL EVALUATION

In our empirical evaluation we seek to answer the following questions:

Q1 Do proximity-based or structural embedding methods adapt better to the temporal link prediction task?
Q2 Does embedding alignment improve the performance of proximity-based and structural embedding methods?
Q3 How should node embeddings be combined across time?
Q4 What edge operators lead to better performance in temporal settings?
Q5 How stable are methods with respect to different design choices or datasets?

Before we answer these questions, we present the datasets and experimental setup that we consider in our evaluation.

5.1 Datasets & Setup

5.1.1 Datasets. We use four real-world datasets that are available at SNAP [17] and/or NetworkRepository [22]. To create a snapshot, we parse the edge-list such that only edge occurring within discrete periods of given granularity are included; e.g., edges between (1/1/1970 - 2/1/1970) form one monthly snapshot. The chosen temporal granularity, and the corresponding number of time snapshots per dataset are shown in Table 2. We choose the corresponding granularity to avoid having many empty snapshots, in favor of fewer denser snapshots.

Table 2: Dataset Information

Name	$\|V\|$	$\|E\|$	T	**Granularity**
Bitcoin	3,783	24,186	63	months
Facebook	899	33,720	24	weeks
Wiki-Elec	7,118	107,071	47	months
Email-EU	987	332,334	17	months

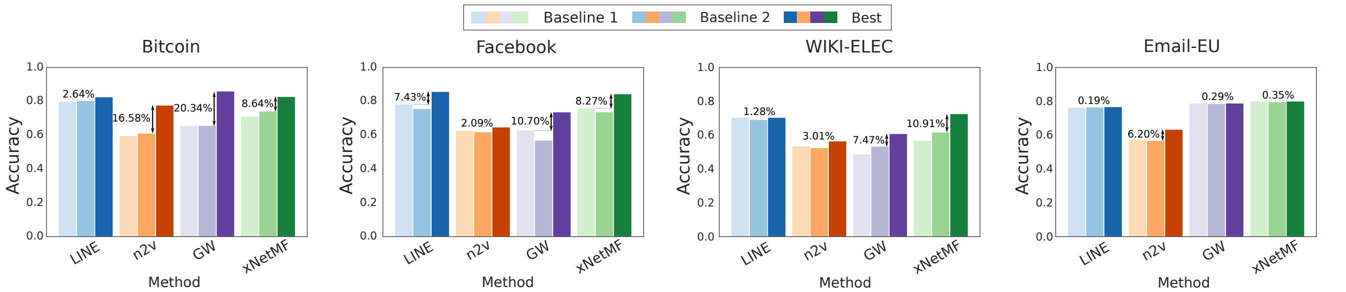

Figure 2: Results of temporal link prediction on four datasets. Percentages over the bar plots show the absolute improvement on the performance of the best performing node alignment, combination and edge generation method over the best-performing baseline.

5.1.2 *Temporal Link Prediction Task.* Following [24] and [6], we evaluate our embeddings on a temporal link prediction task. We generate embeddings using the methods: LINE, node2vec, GraphWave and xNetMF at every snapshot G_i. Given graphs and corresponding node embeddings from timestamps 1 to $T-1$, we build a Logistic Regression model to predict links in the graph at timestamp T. We construct the training set from G_{T-1} by sampling true (existing) and an equal number of false (non-existent) edges. We generate the test set from G_T, using a similar procedure. We report the average accuracy across 5 trials. See Appendix B for detailed results.

5.2 Q1. Link Prediction Performance

The best predictive performance of temporal link prediction per method and dataset is shown in Figure 2. We also include the performance of two baseline combinations of design options **without embedding alignment (P2)** and **without temporal combination (P3)**, which generate edge embeddings using concatenation and averaging, respectively.

First, we observe that each method in every dataset benefits from correctly-picked alignment, temporal combination and edge embedding generation techniques, as evidenced by the improvement over the baselines. The improvement in performance is the most evident in Bitcoin dataset, whereas it is not very significant in Email-EU dataset. We hypothesize that this behavior is due to the fact that Bitcoin has a sharp decrease in edges towards the later time-stamps, whereas Email-EU remains relatively stable.

Typically, proximity-based embeddings are used in link prediction tasks, since they capture homophily and community structure; and links are more likely to form within communities than across them [7, 23]. Hence, our initial hypothesis was that proximity-based methods should benefit more from careful choice of designs. Surprisingly, however, we observe that structural methods GraphWave and xNetMF perform at least as well as proximity-based methods, and in several datasets outperform the latter. We see the steepest rise in the performance in Bitcoin dataset with GraphWave with 20.34% absolute (equivalent to 31% relative) increase.

5.3 Q2-Q5. Effectiveness of Design Options

To examine the effects of each phase in our pipeline individually, we report the number of times each design choice ranked top-3 across all datasets and methods in temporal link prediction task in Tables 3, 4 and 5. Henceforth, we report our findings on these parameter choices.

5.3.1 **Q2.** *Choice of Alignment.* In our experiments, we tried two different versions of Procrustes for embedding alignment purposes (P2). In the first one, we aligned the nodes in all of the snapshots to the first snapshot, whereas we aligned each consecutive snapshots with one another (as proposed in [28]) in the second one. Since there were not considerable differences in the results for these two types of Procrustes, we only report the former version.

Table 3: Number of times each alignment choice ranked top-3 across all datasets.

	Proximity	Structural	Total
No Alignment	9	11	20
Averaging Runs [29]	11	8	19
Procrustes	4	5	9

Firstly, we observe in Table 3 that No Alignment and Averaging Runs performed better than Procrustes on average. Despite its successes in recent research works [9, 28], we did not find it particularly effective when combining static embeddings over time. Averaging Runs [29] is not originally proposed for the alignment task, but it functions as one in this setting, and performs quite well. It is interesting that having No Alignment across timestamps still yields strong results, especially for proximity-based embedding methods that yield embedding spaces that could be rotated, translated, or rescaled relative to each other [4]. That said, we note that the impact of alignment is slightly more pronounced in proximity-based embeddings than structural ones.

5.3.2 **Q3.** *Choice of Temporal Embedding Combination.* We often observe that either No Combination across time, or Linear Combination works better than using Exponential Decay in Table 4. Our initial assumption of "more recent snapshots being more important for link prediction task" did not hold in the case of Exponential Decay. Weighting more recent embeddings exponentially more results in worse predictive performance. We note that in our analysis, Exponential Decay was never in the top-3 combinations for structural node embeddings.

It appears that summing up embeddings across time works very well, especially for structural embedding methods (it ranks in the top-3 in 17/24 cases). On the other hand, proximity-based methods achieve high scores even without utilizing past embedding information. This, of course, depends on the density of the snapshots. When the last snapshot alone is dense enough (e.g. Email-EU), the embeddings in the last snapshot are representative enough for the test snapshot, and this results in high accuracy even without temporal combination. Therefore, it is expected that the effect of temporal combination matters more in more sparse datasets.

Table 4: Number of times each temporal combination technique ranked top-3 across all datasets.

	Proximity	Structural	Total
No Combination	11	7	18
Linear Comb. (Sum)	9	17	26
Exponential Decay	4	0	4

5.3.3 **Q4.** *Edge Generation Method.* For the edge generation step, we observe that concatenation of node embeddings is the best performing heuristic in Table 5. Averaging runs follow that, and Hadamard product performs the worst on average. Despite the recommendation of Hadamard product for edge generation in static settings [8], our results show that it is consistently outperformed by concatenation and averaging of node features in temporal settings. The difference between proximity-based and structural embedding methods is more pronounced when Hadamard product is used (see Table 5). There, we see Hadamard product performs especially poorly for proximity-based methods, whereas structural methods (specifically GraphWave) benefits from it. All in all, we recommend the usage of concatenation over the other heuristics for future research.

Table 5: Number of times each edge generation technique ranked top-3 across all datasets.

	Proximity	Structural	Total
Concatenation	15	10	25
Average	9	8	17
Hadamard Prod.	0	6	6

5.3.4 **Q5.** *Embedding Stability.* We find that different datasets lead to different levels of embedding stability, which we define as the variance of average accuracy in the temporal link prediction task across different combinations of design options. Understanding the stability of a method is important so that practitioners can make informed choices when doing hyper-parameter tuning—i.e., should they expect to see better performance with different design choices or select a different method.

In Figure 3, we compute the "distribution" over all the Temporal Combinations × Alignment × Edge Operators options considered for each of methods, for every dataset. We see that some datasets

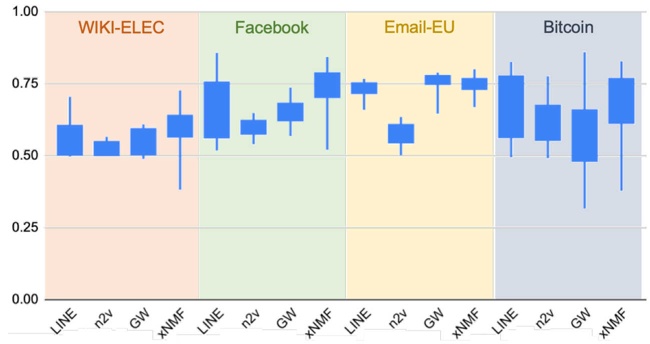

Figure 3: Distribution of accuracy across each method and dataset (over different combinations of design options).

appear tricky for all methods. For example, for Bitcoin dataset, we see relatively wide range of values across all combinations. On the contrary, Email-EU appears to be more stable, with LINE, GraphWave and xNetMF performing comparably. Even though node2vec performs relatively poorly, it still exhibits moderate variance.

However, if we consider performance for a given method, we see that LINE benefits greatly from sensible choices on 3 out of 4 datasets. Node2vec does not see as much benefit, and tends to perform poorly regardless (with the exception being Bitcoin). Graphwave has different performance depending on dataset; having considerable variance on Bitcoin, but relatively less on Email-EU. Regardless, it too is benefited by good design choices. xNetMF also has large variance on 3 of 4 datasets, but it produces the best or near best accuracy when properly tuned.

6 CONCLUSION

In this work, we investigate various design choices that need to be considered when extending structural and proximity-based embeddings to dynamic settings. Specifically, we consider different embedding alignment strategies, temporal combinations schemes, and edge operators. We find that selecting good choices may result in up to 20% increase over baselines without aligning embeddings nor combining embeddings from different snapshots. Moreover, when selecting alignment methods, No-Combination or Averaging are preferred. Either of No Combination and Linear Combination (summation) over time-steps is preferred for incorporating temporal dynamics. Additionally, we find that using the Hadamard product is generally not effective for generating edge embeddings for the link prediction task. We hope that this guidance on design choices may be valuable to practitioners and the research community when designing baselines that can be used to evaluate the performance of new dynamic embedding methods.

ACKNOWLEDGEMENTS

We would like to thank the reviewers for their feedback and suggestions for future directions. This work is supported by the NSF under Grant No. IIS 1845491, Army Young Investigator Award No. W911NF1810397, and Adobe, Amazon, and Google faculty awards.

REFERENCES

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery* 29, 3 (2015), 626–688.

[2] Toine Bogers. 2010. Movie recommendation using random walks over the contextual graph. In *Proc. of the 2nd Intl. Workshop on Context-Aware Recommender Systems*.

[3] Xiyuan Chen, Mark Heimann, Fatemeh Vahedian, and Danai Koutra. 2020. Consistent Network Alignment via Proximity-Preserving Node Embedding. In *arXiv:2005.04725*.

[4] Xiyuan Chen, Mark Heimann, Fatemeh Vahedian, and Danai Koutra. 2020. Consistent Network Alignment with Node Embedding. *arXiv preprint arXiv:2005.04725* (2020).

[5] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. 2018. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1320–1329.

[6] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2019. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems* (2019), 104816.

[7] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.

[8] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM KDD*. ACM, 855–864.

[9] William L Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic word embeddings reveal statistical laws of semantic change. *arXiv preprint arXiv:1605.09096* (2016).

[10] Mark Heimann and Danai Koutra. 2017. On Generalizing Neural Node Embedding Methods to Multi-Network Problems. In *KDD MLG Workshop*.

[11] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. 2018. Regal: Representation learning-based graph alignment. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 117–126.

[12] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. 2012. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1231–1239.

[13] Bernard J Jansen and Soo Young Rieh. 2010. The seventeen theoretical constructs of information searching and information retrieval. *JASIST* 61, 8 (2010), 1517–1534.

[14] Di Jin, Mark Heimann, Ryan Rossi, and Danai Koutra. 2019. node2bits: Compact Time-and Attribute-aware Node Representations. In *ECML/PKDD European Conference on Principles and Practice of Knowledge Discovery in Databases*.

[15] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. 2013. Deltacon: A principled massive-graph similarity function. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 162–170.

[16] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM KDD*. ACM, 1269–1278.

[17] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford large network dataset collection.

[18] Weiping Liu and Linyuan Lü. 2010. Link prediction based on local random walk. *EPL (Europhysics Letters)* 89, 5 (2010), 58007.

[19] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*. International World Wide Web Conferences Steering Committee, 969–976.

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM KDD*. ACM, 701–710.

[21] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 459–467.

[22] Ryan Rossi and Nesreen Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[23] Ryan A. Rossi, Di Jin, Sungchul Kim, Nesreen Ahmed, Danai Koutra, and John Boaz Lee. 2020. From Community to Role-based Graph Embeddings. *ACM Transactions on Knowledge Discovery from Data (TKDD)* (2020).

[24] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 519–527.

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

[26] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. 2015. Timecrunch: Interpretable dynamic graph summarization. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1055–1064.

[27] Umang Sharan and Jennifer Neville. 2008. Temporal-relational classifiers for prediction in evolving domains. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 540–549.

[28] Uriel Singer, Ido Guy, and Kira Radinsky. 2019. Node Embedding over Temporal Graphs. *arXiv preprint arXiv:1903.08889* (2019).

[29] Balasubramaniam Srinivasan and Bruno Ribeiro. 2019. On the Equivalence between Node Embeddings and Structural Graph Representations. *arXiv preprint arXiv:1910.00452* (2019).

[30] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[31] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.

[32] Yu Xie, Chunyi Li, Bin Yu, Chen Zhang, and Zhouhua Tang. 2020. A Survey on Dynamic Network Embedding. arXiv:2006.08093 [cs.SI]

[33] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding Temporal Network via Neighborhood Formation. In *Proceedings of the 24th ACM KDD* (London, United Kingdom) *(KDD '18)*. ACM, New York, NY, USA, 2857–2866. https://doi.org/10.1145/3219819.3220054

A HYPERPARAMETER TUNING

We used the default parameters for all methods. Specifically,

- Node2Vec: We used $p = 4$ and $q = 1$.
- GraphWave: We used the provided "auto" select option for τ.
- LINE: We used second-order proximity, $K = 5$, $lr = 0.025$, $batches = 5000$, and $batch - size = 128$.
- xNetMF: We used $alpha = 0.1$, $k = 1$, $gamma - struc = 1$, $gamma - att = 1$.

For each method, we replicated the embeddings 6 times. We sampled 3 replicates from these 6 embeddings to compute the average alignment embeddings. We fixed the embedding selections, and trained the classifier 3 times. We repeat this procedure 3 times to get the reported average accuracy and standard deviation. We sample true edges from G_{T-1} and an equivalent number of false edges to train the classifier. We generate the test similarly using G_T.

We use the following popular, publicly available implementations, provided by the respective authors:

- https://github.com/aditya-grover/node2vec (Node2Vec)
- https://github.com/snowkylin/line (LINE)
- https://github.com/snap-stanford/graphwave/ (GraphWave)
- https://github.com/GemsLab/REGAL (xNETMF)

B DETAILED RESULTS

Average accuracy and standard deviation of all combinations of methods across all datasets are shown in Tables 6, 7, 8 and 9.

Table 6: Wiki-Elec: Average accuracy and standard deviation per method for the different combinations of design options.

Combination	Edge Operator	Alignment	LINE	node2vec	GraphWave	XNetMF
Exponential Decay	Average	None	0.548±0.015	0.556±0.040	0.521±0.003	0.571±0.028
		Averaging	0.533±0.018	0.563±0.031	0.522±0.004	0.566±0.028
		Procrustes	0.550±0.035	0.531±0.024	0.504±0.004	0.557±0.021
	Concat	None	0.545±0.019	0.561±0.045	0.528±0.003	0.584±0.036
		Averaging	0.525±0.034	0.565±0.043	0.529±0.002	0.578±0.035
		Procrustes	0.553±0.042	0.538±0.039	0.504±0.004	0.567±0.028
	Hadamard	None	0.504±0.007	0.502±0.007	0.533±0.005	0.560±0.025
		Averaging	0.498±0.007	0.501±0.007	0.533±0.006	0.559±0.024
		Procrustes	0.502±0.007	0.504±0.006	0.500±0.001	0.518±0.060
No Combination	Average	None	0.692±0.014	0.526±0.016	0.534±0.058	0.617±0.037
		Averaging	0.688±0.006	0.521±0.007	0.531±0.071	0.640±0.054
		Procrustes	0.688±0.015	0.506±0.005	0.535±0.065	0.637±0.046
	Concat	None	0.705±0.016	0.535±0.021	0.489±0.030	0.568±0.047
		Averaging	0.703±0.008	0.524±0.007	0.499±0.003	0.558±0.052
		Procrustes	0.693±0.019	0.502±0.007	0.492±0.029	0.589±0.063
	Hadamard	None	0.512±0.009	0.501±0.004	0.544±0.058	0.593±0.025
		Averaging	0.501±0.013	0.500±0.003	0.545±0.061	0.622±0.026
		Procrustes	0.505±0.011	0.500±0.003	0.500±0.000	0.383±0.119
Linear Combination	Average	None	0.590±0.029	0.556±0.040	0.609±0.035	0.691±0.033
		Averaging	0.569±0.014	0.543±0.021	0.608±0.037	0.704±0.040
		Procrustes	0.595±0.022	0.530±0.025	0.597±0.098	0.717±0.061
	Concat	None	0.575±0.038	0.555±0.048	0.593±0.036	0.628±0.042
		Averaging	0.560±0.008	0.550±0.038	0.591±0.035	0.635±0.089
		Procrustes	0.611±0.031	0.537±0.039	0.599±0.100	0.727±0.054
	Hadamard	None	0.497±0.006	0.503±0.007	0.608±0.037	0.682±0.029
		Averaging	0.497±0.006	0.501±0.007	0.609±0.036	0.687±0.028
		Procrustes	0.504±0.007	0.504±0.007	0.506±0.009	0.602±0.085

Table 7: Facebook: Average accuracy and standard deviation per method for the different combinations of design options.

Combination	Edge Operator	Alignment	LINE	node2vec	GraphWave	XNetMF
Exp Decay	Average	None	0.619±0.055	0.599±0.032	0.674±0.026	0.698±0.030
		Averaging	0.582±0.040	0.577±0.022	0.664±0.026	0.716±0.041
		Procrustes	0.656±0.030	0.597±0.036	0.684±0.027	0.711±0.036
	Concat	None	0.673±0.036	0.621±0.027	0.672±0.020	0.729±0.029
		Averaging	0.649±0.018	0.629±0.041	0.661±0.038	0.732±0.037
		Procrustes	0.757±0.044	0.607±0.031	0.670±0.028	0.776±0.031
	Hadamard	None	0.559±0.054	0.555±0.024	0.672±0.032	0.637±0.040
		Averaging	0.518±0.039	0.577±0.028	0.688±0.029	0.604±0.028
		Procrustes	0.521±0.039	0.554±0.042	0.572±0.012	0.546±0.029
No Combination	Average	None	0.757±0.034	0.620±0.036	0.570±0.007	0.738±0.022
		Averaging	0.744±0.022	0.621±0.029	0.574±0.012	0.738±0.018
		Procrustes	0.753±0.031	0.620±0.038	0.625±0.009	0.732±0.019
	Concat	None	0.784±0.015	0.627±0.038	0.630±0.006	0.761±0.016
		Averaging	0.786±0.017	0.626±0.020	0.627±0.010	0.765±0.015
		Procrustes	0.778±0.018	0.620±0.028	0.631±0.008	0.769±0.012
	Hadamard	None	0.560±0.037	0.556±0.046	0.569±0.013	0.663±0.046
		Averaging	0.561±0.040	0.592±0.026	0.571±0.019	0.723±0.010
		Procrustes	0.555±0.056	0.540±0.038	0.621±0.026	0.521±0.113
Linear Combination	Average	None	0.705±0.036	0.608±0.029	0.672±0.029	0.791±0.034
		Averaging	0.657±0.047	0.600±0.039	0.657±0.030	0.795±0.020
		Procrustes	0.776±0.032	0.628±0.034	0.678±0.025	0.791±0.028
	Concat	None	0.749±0.040	0.648±0.027	0.691±0.021	0.823±0.039
		Averaging	0.736±0.037	0.636±0.021	0.687±0.023	0.838±0.037
		Procrustes	0.858±0.027	0.608±0.029	0.737±0.026	0.844±0.023
	Hadamard	None	0.566±0.049	0.553±0.042	0.707±0.024	0.783±0.032
		Averaging	0.558±0.040	0.583±0.051	0.716±0.031	0.794±0.020
		Procrustes	0.580±0.052	0.553±0.030	0.568±0.014	0.623±0.019

Table 8: Email-EU: Average accuracy and standard deviation per method for the different combinations of design options.

Combination	Edge Operator	Alignment	LINE	node2vec	GraphWave	XNetMF
Exp Decay	Average	None	0.726±0.005	0.607±0.008	0.750±0.002	0.742±0.006
		Averaging	0.726±0.002	0.606±0.008	0.749±0.002	0.736±0.002
		Procrustes	0.717±0.003	0.546±0.004	0.749±0.003	0.726±0.003
	Concat	None	0.723±0.005	0.613±0.007	0.751±0.002	0.743±0.005
		Averaging	0.728±0.003	0.613±0.010	0.749±0.002	0.738±0.003
		Procrustes	0.718±0.003	0.547±0.004	0.750±0.002	0.729±0.004
	Hadamard	None	0.712±0.012	0.550±0.005	0.752±0.002	0.705±0.009
		Averaging	0.706±0.003	0.546±0.005	0.751±0.002	0.715±0.004
		Procrustes	0.659±0.015	0.506±0.004	0.646±0.002	0.669±0.004
No Combination	Average	None	0.766±0.003	0.568±0.005	0.786±0.002	0.797±0.003
		Averaging	0.760±0.003	0.587±0.006	0.787±0.003	0.793±0.002
		Procrustes	0.767±0.003	0.552±0.004	0.786±0.002	0.764±0.005
	Concat	None	0.764±0.004	0.572±0.006	0.789±0.003	0.801±0.003
		Averaging	0.756±0.002	0.591±0.005	0.789±0.002	0.796±0.004
		Procrustes	0.768±0.003	0.552±0.004	0.785±0.002	0.766±0.004
	Hadamard	None	0.745±0.007	0.541±0.004	0.775±0.002	0.761±0.008
		Averaging	0.718±0.002	0.540±0.004	0.776±0.003	0.756±0.003
		Procrustes	0.683±0.014	0.502±0.006	0.670±0.002	0.715±0.015
Linear Combination	Average	None	0.749±0.008	0.625±0.008	0.771±0.002	0.773±0.006
		Averaging	0.745±0.005	0.634±0.010	0.774±0.003	0.766±0.003
		Procrustes	0.750±0.003	0.553±0.005	0.768±0.002	0.762±0.002
	Concat	None	0.748±0.007	0.623±0.008	0.774±0.003	0.773±0.006
		Averaging	0.745±0.003	0.634±0.010	0.775±0.002	0.765±0.003
		Procrustes	0.753±0.003	0.552±0.004	0.768±0.003	0.768±0.003
	Hadamard	None	0.733±0.009	0.560±0.007	0.777±0.002	0.749±0.006
		Averaging	0.735±0.002	0.560±0.005	0.777±0.002	0.747±0.004
		Procrustes	0.704±0.004	0.506±0.004	0.656±0.003	0.687±0.004

Table 9: Bitcoin: Average accuracy and standard deviation per method for the different combinations of design options.

Combination	Edge Operator	Alignment	LINE	node2vec	GraphWave	XNetMF
Exp Decay	Average	None	0.704±0.060	0.668±0.045	0.473±0.040	0.578±0.116
		Averaging	0.646±0.074	0.670±0.038	0.479±0.050	0.731±0.080
		Procrustes	0.660±0.056	0.674±0.068	0.509±0.068	0.492±0.065
	Concat	None	0.716±0.063	0.674±0.055	0.486±0.049	0.564±0.100
		Averaging	0.678±0.034	0.652±0.038	0.470±0.048	0.668±0.068
		Procrustes	0.632±0.052	0.637±0.023	0.474±0.049	0.480±0.058
	Hadamard	None	0.495±0.063	0.514±0.047	0.469±0.052	0.513±0.051
		Averaging	0.536±0.067	0.520±0.061	0.465±0.019	0.731±0.083
		Procrustes	0.503±0.080	0.555±0.109	0.516±0.074	0.544±0.078
No Combination	Average	None	0.803±0.054	0.611±0.047	0.657±0.159	0.742±0.097
		Averaging	0.826±0.011	0.573±0.026	0.658±0.164	0.759±0.052
		Procrustes	0.794±0.048	0.633±0.051	0.720±0.103	0.783±0.018
	Concat	None	0.800±0.018	0.597±0.043	0.655±0.157	0.712±0.091
		Averaging	0.808±0.015	0.555±0.040	0.656±0.162	0.689±0.044
		Procrustes	0.789±0.047	0.613±0.047	0.720±0.103	0.782±0.020
	Hadamard	None	0.575±0.058	0.513±0.062	0.648±0.149	0.707±0.096
		Averaging	0.586±0.034	0.520±0.029	0.648±0.154	0.801±0.030
		Procrustes	0.554±0.057	0.491±0.061	0.317±0.088	0.379±0.211
Linear Combination	Average	None	0.746±0.046	0.759±0.043	0.632±0.103	0.828±0.057
		Averaging	0.788±0.048	0.777±0.033	0.664±0.055	0.775±0.058
		Procrustes	0.763±0.032	0.701±0.040	0.676±0.055	0.708±0.127
	Concat	None	0.734±0.049	0.749±0.040	0.644±0.082	0.820±0.074
		Averaging	0.735±0.052	0.723±0.076	0.627±0.079	0.711±0.055
		Procrustes	0.719±0.060	0.686±0.021	0.627±0.081	0.694±0.106
	Hadamard	None	0.504±0.048	0.584±0.044	0.861±0.052	0.814±0.071
		Averaging	0.526±0.060	0.555±0.053	0.858±0.054	0.703±0.079
		Procrustes	0.524±0.051	0.540±0.059	0.637±0.101	0.650±0.130