DYNAMIC GRAPH REPRESENTATION LEARNING VIA SELF-ATTENTION NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning latent representations of nodes in graphs is an important and ubiquitous task with widespread applications such as link prediction, node classification, and graph visualization. Previous methods on graph representation learning mainly focus on static graphs, however, many real-world graphs are dynamic and evolve over time. In this paper, we present Dynamic Self-Attention Network (DySAT), a novel neural architecture that operates on dynamic graphs and learns node representations that capture both structural properties and temporal evolutionary patterns. Specifically, DySAT computes node representations by jointly employing self-attention layers along two dimensions: structural neighborhood and temporal dynamics. We conduct link prediction experiments on two classes of graphs: communication networks and bipartite rating networks. Our experimental results show that DySAT has a significant performance gain over several different state-of-the-art graph embedding baselines.

1 INTRODUCTION

Learning latent representations (or embeddings) of nodes in graphs has been recognized as a fundamental learning problem due to its widespread use in various domains such as social media (Perozzi et al., 2014), biology (Grover & Leskovec, 2016), and knowledge bases (Wang et al., 2014). The basic idea is to learn a low-dimensional vector for each node, which encodes the structural properties of a node and its neighborhood (and possibly attributes). Such low-dimensional representations can benefit a plethora of graph analytical tasks such as node classification, link prediction, and graph visualization (Perozzi et al., 2014; Tang et al., 2015; Grover & Leskovec, 2016; Wang et al., 2016).

Previous work on graph representation learning mainly focuses on static graphs, which contain a fixed set of nodes and edges. However, many graphs in real-world applications are intrinsically dynamic, in which graph structures can evolve over time. They are usually represented as a sequence of graph snapshots from different time steps (Leskovec et al., 2007). Examples include academic co-authorship networks where authors may periodically switch their collaboration behaviors and email communication networks whose structures may change dramatically due to sudden events. In such scenarios, modeling temporal evolutionary patterns is important in accurately predicting node properties and future links.

Learning dynamic node representations is challenging, compared to static settings, due to the complex time-varying graph structures: nodes can emerge and leave, links can appear and disappear, and communities can merge and split. This requires the learned embeddings not only to preserve structural proximity of nodes, but also to jointly capture the temporal dependencies over time. Though some recent work attempts to learn node representations in dynamic graphs, they mainly impose a temporal regularizer to enforce smoothness of the node representations from adjacent snapshots (Zhu et al., 2016; Li et al., 2017; Zhou et al., 2018). However, these approaches may fail when nodes exhibit significantly distinct evolutionary behaviors. Trivedi et al. (2017) employ a recurrent neural architecture for temporal reasoning in multi-relational knowledge graphs. However, their temporal node representations are limited to modeling first-order proximity, while ignoring the structure of higher-order graph neighborhoods.

Attention mechanisms have recently achieved great success in many sequential learning tasks such as machine translation (Bahdanau et al., 2015) and reading comprehension (Yu et al., 2018). The

key underlying principle is to learn a function that aggregates a variable-sized input, while focusing on the parts most relevant to a certain context. When the attention mechanism uses a single sequence as both the inputs and the context, it is often called *self-attention*. Though attention mechanisms were initially designed to facilitate Recurrent Neural Networks (RNNs) to capture long-term dependencies, recent work by Vaswani et al. (2017) demonstrates that a fully self-attentional network itself can achieve state-of-the-art performance in machine translation tasks. Velickovic et al. (2018) extend self-attention to graphs by enabling each node to attend over its neighbors, achieving state-of-the-art results for semi-supervised node classification tasks in static graphs.

As dynamic graphs usually include periodical patterns such as recurrent links or communities, attention mechanisms are capable of utilizing information about most relevant historical context, to facilitate future prediction. Inspired by recent work on attention techniques, we present a novel neural architecture named Dynamic Self-Attention Network (DySAT) to learn node representations on dynamic graphs. Specifically, we employ self-attention along two dimensions: structural neighborhoods and temporal dynamics, *i.e.*, DySAT generates a dynamic representation for a node by considering both its neighbors and historical representations, following a self-attentional strategy. Unlike static graph embedding methods that focus entirely on preserving structural proximity, we learn dynamic node representations that reflect the temporal evolution of graph structure over a varying number of historical snapshots. In contrast to temporal smoothness-based methods, DySAT learns attention weights that capture temporal dependencies at a fine-grained node-level granularity.

We evaluate our framework on the dynamic link prediction task using four benchmarks of different sizes including two email communication networks (Klimt & Yang, 2004; Panzarasa et al., 2009) and two bipartite rating networks (Harper & Konstan, 2016). Our evaluation results show that DySAT achieves significant improvements (3.6% macro-AUC on average) over several state-of-the-art baselines and maintains a more stable performance over different time steps.

2 Related Work

Our framework is related to previous representation learning techniques on static graphs, dynamic graphs, and recent developments in self-attention mechanisms.

Static graph embeddings. Early work on unsupervised graph representation learning exploits the spectral properties of various graph matrix representations, such as Laplacian, etc. to perform dimensionality reduction (Tenenbaum et al., 2000; Belkin & Niyogi, 2001). To improve scalability, some work (Perozzi et al., 2014; Grover & Leskovec, 2016) utilizes Skip-gram methods, inspired by their success in Natural Language Processing (NLP). Recently, several graph neural network architectures based on generalizations of convolutions have achieved tremendous success, among which many methods are designed for supervised or semi-supervised learning tasks (Niepert et al., 2016; Defferrard et al., 2016; Kipf & Welling, 2017; Sankar et al., 2017; Velickovic et al., 2018). Hamilton et al. (2017) extend graph convolutional methods through trainable neighborhood aggregation functions, to propose a general framework applicable to unsupervised representation learning. However, these methods are not designed to model temporal evolutionary patterns in dynamic graphs.

Dynamic graph embeddings. Most techniques employ temporal smoothness regularization to ensure embedding stability across consecutive time-steps (Zhu et al., 2016; Li et al., 2017). Zhou et al. (2018) additionally use triadic closure (Kossinets & Watts, 2006) as guidance, leading to significant improvements. Neural methods were recently explored in the knowledge graph domain by Trivedi et al. (2017), who employ a recurrent neural architecture for temporal reasoning. However, their model is limited to tracing link evolution, thus limited to capturing first-order proximity. Goyal et al. (2017) learn incremental node embeddings through initialization from the previous time steps, however, this may not guarantee the model to capture long-term graph similarity. Unlike previous approaches, our framework learns adaptive temporal evolution patterns at a node-level granularity, by capturing the most relevant historical contexts through a self-attentional architecture.

Self-attention mechanisms. Recent advancements in many NLP tasks have demonstrated the superiority of *self-attention* in achieving state-of-the-art performance (Vaswani et al., 2017; Lin et al., 2017; Tan et al., 2018; Shen et al., 2018; Shaw et al., 2018). In DySAT, we employ self-attention mechanisms to compute a dynamic node representation by attending over its neighbors and previous historical representations. Our approach of using self-attention over neighbors is closely related to

the Graph Attention Network (GAT) (Velickovic et al., 2018), which employs neighborhood attention for semi-supervised node classification in a static graph. As dynamic graphs usually contain periodical patterns, we extend the self-attention mechanisms over the historical representations of a particular node to capture its temporal evolution behaviors.

3 Problem Definition

In this work, we address the problem of dynamic graph representation learning. A dynamic graph is defined as a series of observed snapshots, $\mathbb{G} = \{\mathcal{G}^1, \ldots, \mathcal{G}^T\}$ where T is the number of time steps. Each snapshot $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}^t)$ is a weighted undirected graph with a shared node set $\mathcal{V}$, a link set $\mathcal{E}^t$, and weighted adjacency matrix $\boldsymbol{A}^t$ at time t. Unlike some previous work that assumes links can only be added over time in dynamic works, we also allow to remove links. Dynamic graph representation learning aims to learn latent representations $\boldsymbol{e}_v^t \in \mathbb{R}^d$ for each node $v \in \mathcal{V}$ at time steps $t = 1, 2, \ldots, T$, such that $\boldsymbol{e}_v^t$ preserves both the local graph structures centered at v and its evolutionary behaviors prior to time t.

4 Dynamic Self-Attention Network

In this section, we first describe the high-level structure of our model. DySAT consists of two major novel components: *structural* and *temporal self-attention* layers, which can be utilized to construct arbitrary graph neural architectures through stacking of layers. Similar to existing studies on attention mechanisms, we employ multi-head attentions to improve model capacity and stability.

DySAT consists of a *structural* block followed by a *temporal* block, as illustrated in Figure 1, where each block may contain multiple stacked layers of the corresponding layer type. The *structural* block extracts features from the local neighborhood through self-attentional aggregation, to compute intermediate node representations for each snapshot. These representations feed as input to the *temporal* block, which attends over multiple time steps, capturing temporal variations in the graph.

4.1 Structural self-attention

The input of this layer is a graph snapshot $\mathcal{G} \in \mathbb{G}$ and a set of input node representations $\{\boldsymbol{x}_v \in \mathbb{R}^D, \forall v \in \mathcal{V}\}$ where D is the input embedding dimension. The input to the initial layer can be set as 1-hot encoded vectors for each node (or attributes if available). The output is a new set of node representations $\{\boldsymbol{z}_v \in \mathbb{R}^F, \forall v \in \mathcal{V}\}$ with F dimensions that capture local structural properties.

Specifically, the *structural* self-attention layer attends over the immediate neighbors of a node v (in snapshot $\mathcal{G}$), by computing attention weights as a function of their input node embeddings. The structural attention layer is a variant of GAT (Velickovic et al., 2018), applied on a single snapshot:

$$\boldsymbol{z}_v = \sigma\Big(\sum_{u \in \mathcal{N}_v} \alpha_{uv} \boldsymbol{W}^s \boldsymbol{x}_u\Big), \quad \alpha_{uv} = \frac{\exp\Big(\sigma\Big(A_{uv} \cdot \boldsymbol{a}^T[\boldsymbol{W}^s \boldsymbol{x}_u || \boldsymbol{W}^s \boldsymbol{x}_v]\Big)\Big)}{\sum_{w \in \mathcal{N}_v} \exp\Big(\sigma\Big(A_{wv} \cdot \boldsymbol{a}^T[\boldsymbol{W}^s \boldsymbol{x}_w || \boldsymbol{W}^s \boldsymbol{x}_v]\Big)\Big)} \tag{1}$$

where $\mathcal{N}_v = \{u \in \mathcal{V} : (u, v) \in \mathcal{E}\}$ is the set of immediate neighbors of node v in snapshot $\mathcal{G}$; $\boldsymbol{W}^s \in \mathbb{R}^{D \times F}$ is a shared weight transformation applied to each node in the graph; $\boldsymbol{a} \in \mathbb{R}^{2D}$ is a weight vector parameterizing the attention function implemented as feed-forward layer; $||$ is the concatenation operation and $\sigma(\cdot)$ is a non-linear activation function. Note that A_{uv} is the weight of link (u, v) in the current snapshot $\mathcal{G}$. The set of learned coefficients α_{uv}, obtained by a softmax over the neighbors of each node, indicate the importance or contribution of node u to node v at the current snapshot. We use a LeakyRELU non-linearity to compute the attention weights, followed by ELU for the output representations. In our experiments, we employ sparse matrices to implement the *masked* self-attention over neighbors.

4.2 Temporal self-attention

To further capture temporal evolutionary patterns in a dynamic network, we design a temporal self-attention layer. The input of this layer is a sequence of representations of a particular node v at

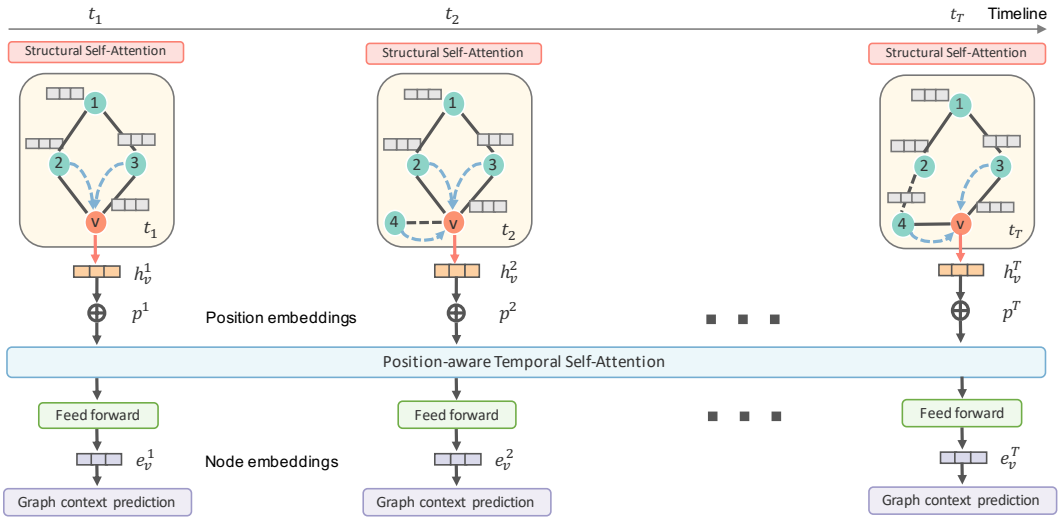

Figure 1: Neural architecture of DySAT: we employ structural attention layers followed by temporal attention layers. Dashed black arrows indicate new links and dashed blue arrows refer to neighbor-based structural-attention.

different time steps. Specifically, for each node v, we define the input as $\{\boldsymbol{x}_v^1, \boldsymbol{x}_v^2, \ldots, \boldsymbol{x}_v^T\}, \boldsymbol{x}_v^t \in \mathbb{R}^{D'}$ where T is the number of time steps and D' is the dimensionality of the input representations. The layer output is a new representation sequence for v at each time step, *i.e.*, $\boldsymbol{z}_v = \{\boldsymbol{z}_v^1, \boldsymbol{z}_v^2, \ldots, \boldsymbol{z}_v^T\}, \boldsymbol{z}_v^t \in \mathbb{R}^{F'}$ with dimensionality F'. We denote the input and output representations of v, packed together across time, by matrices $\boldsymbol{X}_v \in \mathbb{R}^{T \times D'}$ and $\boldsymbol{Z}_v \in \mathbb{R}^{T \times F'}$ respectively.

The key objective of the temporal self-attentional layer is to capture the temporal variations in graph structure over multiple time steps. The input representation of node v at time-step t, $\boldsymbol{x}_v^t$, constitutes an encoding of the current local structure around v. We use $\boldsymbol{x}_v^t$ as the query to attend over its historical representations ($< t$), tracing the evolution of the local neighborhood around v. Thus, temporal self-attention facilitates learning of dependencies between various representations of a node across different time steps.

To compute the output representation of node v at t, we use the scaled dot-product form of attention (Vaswani et al., 2017) where the queries, keys, and values are set as the input node representations. The queries, keys, and values are first transformed to a different space by using linear projections matrices $\boldsymbol{W}_q \in \mathbb{R}^{D' \times F'}, \boldsymbol{W}_k \in \mathbb{R}^{D' \times F'}$ and $\boldsymbol{W}_v \in \mathbb{R}^{D' \times F'}$ respectively. Here, we allow each time-step t to attend over all time-steps up to and including t, to prevent leftward information flow and preserve the auto-regressive property. The temporal self-attention is defined as:

$$\boldsymbol{Z}_v = \boldsymbol{\beta_v}(\boldsymbol{X}_v \boldsymbol{W}_v), \quad \beta_v^{ij} = \frac{\exp(e_v^{ij})}{\sum_{k=1}^{T} \exp(e_v^{ik})}, \quad e_v^{ij} = \left(\frac{((\boldsymbol{X}_v \boldsymbol{W}_q)(\boldsymbol{X}_v \boldsymbol{W}_k)^T)_{ij}}{\sqrt{F'}} + M_{ij} \right) \tag{2}$$

where $\boldsymbol{\beta_v} \in \mathbb{R}^{T \times T}$ is the attention weight matrix obtained by the multiplicative attention function and $\boldsymbol{M} \in \mathbb{R}^{T \times T}$ is a mask matrix with each entry $M_{ij} \in \{-\infty, 0\}$. When $M_{ij} = -\infty$, the softmax function results in a zero attention weight, *i.e.*, $\beta_v^{ij} = 0$, which switches off the attention from time-step i to j. To encode the temporal order, we define $\boldsymbol{M}$ as:

$$M_{ij} = \begin{cases} 0, & i \leq j \\ -\infty, & \text{otherwise} \end{cases}$$

4.3 Multi-Head Attention

We additionally employ multi-head attention (Vaswani et al., 2017) to jointly attend to different subspaces at each input, leading to a leap in model capacity. We use multiple attention heads,

	Communication Networks		Rating Networks	
Dataset	**Enron**	**UCI**	**Yelp**	**ML-10M**
# Nodes	143	1,809	6,569	20,537
# Links	2,347	16,822	95,361	43,760
# Time steps	10	13	12	13

Table 1: Statistics of the datasets used in our experiments

followed by concatenation, in both structural and temporal self-attention layers:

$$\text{Structural multi-head self-attention:} \qquad \boldsymbol{h}_v = \text{Concat}(\boldsymbol{z}_v^1, \boldsymbol{z}_v^2, \ldots, \boldsymbol{z}_v^H) \qquad \forall v \in V \tag{3}$$

$$\text{Temporal multi-head self-attention:} \qquad \boldsymbol{H}_v = \text{Concat}(\boldsymbol{Z}_v^1, \boldsymbol{Z}_v^2, \ldots, \boldsymbol{Z}_v^H) \qquad \forall v \in V \tag{4}$$

where H is the number of attention heads, $\boldsymbol{h}_v \in \mathbb{R}^F$ and $\boldsymbol{H}_v \in \mathbb{R}^{T \times F'}$ are the outputs of structural and temporal multi-head attentions respectively. Note that while structural attention is applied on a single snapshot, temporal attention operates over multiple time-steps.

4.4 DySAT Architecture

In this section, we present our neural architecture DySAT for Dynamic Graph Representation Learning, that uses the above defined *structural* and *temporal* self-attention layers as fundamental modules. As shown in Figure 1, DySAT has three modules from its top to bottom, (1) *structural* attention block, (2) *temporal* attention block, and (3) graph context prediction. The model takes as input a collection of T graph snapshots, and generates outputs latent node representations at each time step.

Structural attention block. This module is composed of multiple stacked structural self-attention layers to extract features from nodes at different distances. We apply each layer independently at different snapshots with shared parameters, as illustrated in Figure 1, to capture local neighborhood structure around a node at each time step. However, it should be noted that the embeddings input to a layer can potentially vary across different snapshots. We denote the node representations output by the structural attention block, as $\{\boldsymbol{h}_v^1, \boldsymbol{h}_v^2, \ldots, \boldsymbol{h}_v^T\}, \boldsymbol{h}_v^t \in \mathbb{R}^f$, which feed as input to the *temporal* attention block.

Temporal attention block. First, we equip the temporal attention module with a sense of ordering through *position* embeddings (Gehring et al., 2017), $\{\boldsymbol{p}^1, \ldots, \boldsymbol{p}^T\}, \boldsymbol{p}^t \in \mathbb{R}^f$, which embed the absolute temporal position of each snapshot. The position embeddings are then combined with the output of the structural attention block to obtain a sequence of input representations: $\{\boldsymbol{h}_v^1 + \boldsymbol{p}^1, \boldsymbol{h}_v^2 + \boldsymbol{p}^2, \ldots, \boldsymbol{h}_v^T + \boldsymbol{p}^T\}$ for node v across multiple time steps. This block also follows a similar structure with multiple stacked temporal self-attention layers. The outputs of the final layer pass into a position-wise *feed-forward* layer to give the final node representations $\{\boldsymbol{e}_v^1, \boldsymbol{e}_v^2, \ldots, \boldsymbol{e}_v^T\}$ $\forall v \in V$.

Graph context prediction. To ensure that the learned representations capture both structural and temporal information, we define an objective function that preserves the local structure around a node, across multiple time steps. We use the dynamic representations of a node v at time step t, $\boldsymbol{e}_v^t$ to predict the occurrence of nodes appearing the local neighborhood around v at t. In particular, we use a binary cross-entropy loss function at each time step to encourage nodes co-occurring in fixed-length random walks, to have similar representations.

$$L_v = \sum_{t=1}^{T} \sum_{u \in \mathcal{N}_{walk}^t(v)} -\log(\sigma(< \boldsymbol{e}_u^t, \boldsymbol{e}_v^t >)) - w_n \cdot \sum_{u' \in P_n^t(v)} \log(1 - \sigma(< \boldsymbol{e}_{u'}^t, \boldsymbol{e}_v^t >)) \tag{5}$$

where σ is the sigmoid function, $< . >$ denotes the inner product operation, $\mathcal{N}_{walk}^t(v)$ is the set of nodes that co-occur with v on fixed-length random walks at snapshot t, P_n^t is a negative sampling distribution for snapshot $\mathcal{G}^t$, and w_n, negative sampling ratio, is a tunable hyper-parameter to balance the positive and negative samples.

5 EXPERIMENTS

We evaluate the quality of our learned node representations on the fundamental task of dynamic link prediction. We choose this task since it has been widely used (Trivedi et al., 2017; Goyal et al., 2017; Li et al., 2018) in evaluating the quality of dynamic node representations to predict the temporal evolution in graph structure.

In our experiments, we compare the performance of DySAT against a variety of static and dynamic graph representation learning baselines. Our experimental results on four publicly available benchmarks indicate that DySAT achieves significant performance gain than other models.

5.1 DATASETS

We use four dynamic graph datasets with two communication and bipartite rating networks each.

Communication networks. We consider two publicly available communication network datasets: Enron (Klimt & Yang, 2004) and UCI (Panzarasa et al., 2009). In Enron, the communication links are email interactions between core employees and the links in UCI represent messages sent between users on an online social network platform.

Rating networks. We use two bipartite rating networks from Yelp[1] and MovieLens (Harper & Konstan, 2016). In Yelp, the dynamic graph comprises links between two types of nodes, users and businesses, derived from the observed ratings over time. ML-10M consists of a user-tag interaction network where user-tag links connects users with the tags they applied on certain movies.

In each dataset, multiple graph snapshots are created based on the observed interactions in fixed-length time windows. Table 1 illustrates the statistics of the different datasets and additional details are provided in Appendix D.

5.2 EXPERIMENTAL SETUP

We conduct experiments on the task of link prediction in dynamic graphs, where we learn dynamic node representations on snapshots $\{\mathcal{G}^1, \ldots, \mathcal{G}^t\}$ and use $\{\boldsymbol{e}_v^t, \forall v \in \mathcal{V}\}$ to predict the links at $\mathcal{G}^{t+1}$. We compare different models based on their ability to correctly classify each example (pairs of nodes) into links and non-links. We create examples from the links in $\mathcal{G}^{t+1}$ and an equal number of randomly sampled pairs of unconnected nodes (non-links). We randomly sample 20% of the examples for training and another 20% of the examples as a validation set to tune the hyper-parameters across all models. The remaining 60% of the examples are used as the test set. We repeat this for 10 randomized runs and report the average performance in our results.

We evaluate the performance of different models by training a logistic regression classifier for dynamic link prediction (Zhou et al., 2018). We follow the strategy recommended by Grover & Leskovec (2016) to compute the feature representation for a pair of nodes, using the Hadamard Operator ($\boldsymbol{e}_u^t \odot \boldsymbol{e}_v^t$), for all methods unless explicitly specified otherwise. The Hadamard operator computes the element-wise product of two vectors and closely mirrors the widely used inner product operation in learning node embeddings. We evaluate the performance of link prediction using Area Under the ROC Curve (AUC) scores (Grover & Leskovec, 2016). We also report the average precision scores in Table 5 of the Appendix.

We implement DySAT in Tensorflow (Abadi et al., 2016) and employ mini-batch gradient descent with Adam optimizer (Kingma & Ba, 2015) for training. For Enron, we use a single layer in both the structural and temporal blocks, with each layer comprising 16 attention heads computing 8 features apiece (for a total of 128 dimensions). In the other datasets, we use two structural self-attentional layers with 16 and 8 heads respectively, each computing 16 features (layer sizes of 256, 128). The model is trained for a maximum of 200 epochs with a batch size of 256 nodes and the best performing model on the validation set, is chosen for evaluation. In the rating networks, we observe convergence within 50 epochs, due to the larger number of mini-batch updates per epoch. Appendix C contains further details on hyper-parameter settings.

[1] https://www.yelp.com/dataset/challenge

Method	Enron		UCI		Yelp		ML-10M	
	Micro-AUC	**Macro-AUC**	**Micro-AUC**	**Macro-AUC**	**Micro-AUC**	**Macro-AUC**	**Micro-AUC**	**Macro-AUC**
node2vec	83.72 ± 0.7	83.05 ± 1.2	79.99 ± 0.4	80.49 ± 0.6	67.86 ± 0.2	65.34 ± 0.2	87.74 ± 0.2	87.52 ± 0.3
G-SAGE	$82.48^{*} \pm 0.6$	$81.88^{*} \pm 0.5$	$79.15^{*} \pm 0.4$	$82.89^{*} \pm 0.2$	$60.95^{\dagger} \pm 0.1$	$58.56^{\dagger} \pm 0.2$	$86.19^{\ddagger} \pm 0.3$	$89.92^{\ddagger} \pm 0.1$
G-SAGE + GAT	72.52 ± 0.4	73.34 ± 0.6	74.03 ± 0.4	79.83 ± 0.2	66.15 ± 0.1	65.09 ± 0.2	83.97 ± 0.3	84.93 ± 0.1
DynamicTriad	80.26 ± 0.8	78.98 ± 0.9	77.59 ± 0.6	80.28 ± 0.5	63.53 ± 0.3	62.69 ± 0.3	88.71 ± 0.2	88.43 ± 0.1
Know-Evolve	61.57 ± 1.1	62.28 ± 1.5	71.20 ± 0.5	80.93 ± 0.2	56.88 ± 0.2	59.68 ± 0.2	78.80 ± 0.5	83.70 ± 0.2
DynGEM	67.83 ± 0.6	69.72 ± 1.3	77.49 ± 0.3	79.82 ± 0.5	66.02 ± 0.2	65.94 ± 0.2	73.69 ± 1.2	85.96 ± 0.3
DySAT	$\mathbf{85.71} \pm 0.3$	$\mathbf{86.60} \pm 0.2$	$\mathbf{81.03} \pm 0.2$	$\mathbf{85.81} \pm 0.1$	$\mathbf{70.15} \pm 0.1$	$\mathbf{69.87} \pm 0.1$	$\mathbf{90.82} \pm 0.3$	$\mathbf{93.68} \pm 0.1$

Table 2: Experiment results on dynamic link prediction (micro and macro averaged AUC with standard deviation). We show GraphSAGE (denoted by G-SAGE) results with the best performing aggregators for each dataset ($*$ represents GCN, † represents LSTM, and ‡ represents max-pooling).

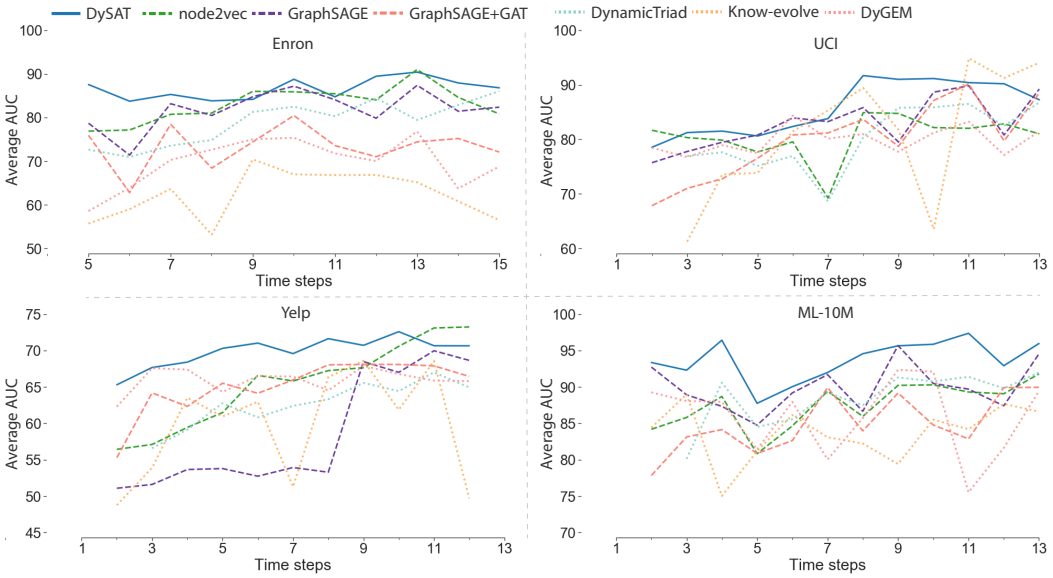

Figure 2: Performance comparison of DySAT with different models across multiple time steps: the solid line represents DySAT; dashed lines represent static graph embedding models; and dotted lines represent dynamic graph embedding models. We truncate the y-axis to avoid visual clutter.

5.3 BASELINE

We compare the performance of DySAT with several state-of-the-art dynamic graph embedding techniques. In addition, we includes several static graph embedding methods in comparison to analyze the gains of using temporal information for dynamic link prediction. To make a fair comparison with static methods, we provide access to the entire history of snapshots by constructing an aggregated graph upto time t, where the weight of each link is defined as the cumulative weight till t agnostic of its occurrence times. We use author-provided implementations for all the baselines and set the final embedding dimension $d = 128$.

We compare against two state-of-the-art unsupervised static embedding methods: node2vec (Grover & Leskovec, 2016) and GraphSAGE (Hamilton et al., 2017). We experiment with different aggregators in GraphSAGE, namely, GCN, mean-pooling, max-pooling, and LSTM, to report the performance of the best performing aggregator in each dataset. To provide a fair comparison with GAT (Velickovic et al., 2018), which originally conduct experiments only on node classification, we implement a graph attention layer (GAT) as an additional aggregator in the framework of GraphSAGE, which we denote by GraphSAGE + GAT. In the dynamic setting, we evaluate DySAT against the most recent studies on dynamic graph embedding including Know-Evolve (Trivedi et al., 2017), DynamicTriad (Zhou et al., 2018), and DynGEM (Goyal et al., 2017). The details of hyperparameter tuning for all methods can be found in Appendix C.

5.4 EXPERIMENTAL RESULTS

We evaluate the models at each time step t by training separate models up to snapshot t and evaluate at $t+1$ for each $t = 1, \ldots, T$. We summarize the micro and macro averaged AUC scores (across all time steps) for all models in Table 2. From the results, we observe that DySAT achieves consistent gains of 3–4% macro-AUC, in comparison to the best baseline across all datasets.

Further, we compare the performance of various models each time step to obtain a deep understanding of their temporal behaviors as illustrated in Figure 2. We observe that the performance of DySAT is relatively more stable than other methods. This contrast is pronounced in the communication networks (Enron and UCI), where we observe drastic drops in performance of GraphSAGE and node2vec at certain time steps.

6 DISCUSSION

Our experimental results provide several interesting observations and insights to the performance of different graph embedding techniques.

First, we observe that GraphSAGE achieves comparable performance to DynamicTriad across different datasets, despite being trained only on static graphs. One possible explanation may be that GraphSAGE uses trainable neighbor-aggregation functions, while DynamicTriad employs Skip-gram based methods augmented with temporal smoothness. This leads us to conjecture that the combination of structural and temporal modeling with expressive aggregation functions, such as multi-head attention, is responsible for the consistently superior performance of DySAT on dynamic link prediction. We also observe that node2vec achieves consistent performance agnostic of temporal information, which demonstrates the effectiveness of second-order random walk sampling. This observation points to the direction of applying sampling techniques to further improve DySAT.

In DySAT, we employ structural attention layers followed by temporal attention layers. We choose this design because graph structures are not stable over time, which makes directly employing structural attention layers after temporal attention layers infeasible. We also consider another alternative design choice that applies self-attention along the two dimensions of neighbors and time together following the strategy similar to (Shen et al., 2018). In practice, this would be computationally expensive due to variable number of neighbors per node across multiple snapshots. We leave exploring other architectural design choices based on structural and temporal self-attentions as future work.

In the current setup, we store the adjacency matrix of each snapshot in memory using sparse matrix, which may pose memory challenges when scaling to large graphs. In the future, we plan to explore DySAT with memory-efficient mini-batch training strategy along the lines of GraphSAGE (Hamilton et al., 2017). Further, we develop an incremental self-attention network (IncSAT) that is efficient in both computation and memory cost as a direct extension of DySAT. Our initial results are promising as reported in Appendix B, which opens the door to future exploration of self-attentional architectures for incremental (or streaming) graph representation learning.

7 CONCLUSION

In this paper, we introduce a novel self-attentional neural network architecture named DySAT to learn node representations in dynamic graphs. Specifically, DySAT computes dynamic node representations using self-attention over the (1) structural neighborhood and (2) historical node representations, thus effectively captures the temporal evolutionary patterns of graph structures. Our experiment results on various real-world dynamic graph datasets indicate that DySAT achieves significant performance gains over several state-of-the-art static and dynamic graph embedding baselines. Though our experiments are conducted on graphs without node features, DySAT can be easily generalized on feature-rich graphs. Another interesting direction is exploring continuous-time generalization of our framework to incorporate more fine-grained temporal variations.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pp. 265–283, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pp. 585–591, 2001.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3837–3845, 2016.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 1243–1252, 2017.

Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. In *IJCAI International Workshop on Representation Learning for Graphs (ReLiG)*, August 2017.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 855–864, 2016.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 1025–1035, 2017.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TIIS)*, 5(4):19, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference for Learning Representations (ICLR)*, 2017.

Bryan Klimt and Yiming Yang. Introducing the enron corpus. In *CEAS 2004 - First Conference on Email and Anti-Spam, July 30-31, 2004, Mountain View, California, USA*, 2004.

Gueorgi Kossinets and Duncan J Watts. Empirical analysis of an evolving social network. *science*, 311(5757):88–90, 2006.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pp. 387–396, 2017.

Jundong Li, Kewei Cheng, Liang Wu, and Huan Liu. Streaming link prediction on dynamic attributed networks. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pp. 369–377, 2018.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. In *International Conference on Learning Representations (ICLR)*, 2017.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 2014–2023, 2016.

Pietro Panzarasa, Tore Opsahl, and Kathleen M. Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *JASIST*, 60(5):911–932, 2009.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pp. 701–710, 2014.

Aravind Sankar, Xinyang Zhang, and Kevin Chen-Chuan Chang. Motif-based convolutional neural network on graphs. *CoRR*, abs/1711.05697, 2017.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pp. 464–468, 2018.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. Disan: Directional self-attention network for rnn/cnn-free language understanding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Zhixing Tan, Mingxuan Wang, Jun Xie, Yidong Chen, and Xiaodong Shi. Deep semantic role labeling with self-attention. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7*, 2018.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pp. 1067–1077, 2015. doi: 10.1145/2736277.2741093.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 3462–3471, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 6000–6010, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1225–1234, 2016. doi: 10.1145/2939672.2939753.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pp. 1112–1119, 2014.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. In *International Conference on Learning Representations (ICLR)*, 2018.

Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7*, 2018.

Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Trans. Knowl. Data Eng.*, 28(10):2765–2777, 2016.

Method	Enron		UCI		Yelp		ML-10M	
	Micro-AUC	Macro-AUC	Micro-AUC	Macro-AUC	Micro-AUC	Macro-AUC	Micro-AUC	Macro-AUC
Original	**85.71** $\pm$ 0.3	**86.60** $\pm$ 0.2	**81.03** $\pm$ 0.2	**85.81** $\pm$ 0.1	**70.15** $\pm$ 0.1	**69.87** $\pm$ 0.1	**90.82** $\pm$ 0.3	**93.68** $\pm$ 0.1
No Temporal	84.50 $\pm$ 0.3	85.68 $\pm$ 0.4	76.61 $\pm$ 0.2	79.97 $\pm$ 0.3	68.34 $\pm$ 0.1	67.20 $\pm$ 0.3	89.61 $\pm$ 0.4	91.10 $\pm$ 0.2

Table 3: Experimental study on removing temporal attention layers from DySAT (micro and macro averaged AUC with standard deviation)

A Effects on Removing Temporal Layers

To demonstrate the effectiveness of temporal self-attention layers, we conduct an experimental study that removes the temporal attention block from DySAT to create a simpler architecture. This model is optimized using the same loss function (Eqn. 5) applied on the intermediate representations $\{\boldsymbol{h}_v^1, \boldsymbol{h}_v^2, \ldots, \boldsymbol{h}_v^T\}$ for each node $v \in \mathcal{V}$. Note that this model is different from static methods since the structural self-attention block is jointly optimized (using Eqn. 5) across all snapshots without any explicit temporal modeling. We use the best configuration of DySAT on each dataset from the original experiments to initialize the new model. The performance comparison is shown in Table 3. We observe that in some datasets, the structural attention block is able to learn some temporal evolution patterns in graph structure, despite the lack of explicit temporal modeling. However, the new model is consistently inferior to DySAT and we observe that the original DySAT has a 3% average gain in Macro-AUC, which validates our choice of using temporal self-attentional layers.

B Incremental Self-Attention Network

In this section, we describe an extension of our dynamic self-attentional architecture to learn incremental node representations. The motivation of incremental learning arises due to the proliferation in sizes of real-world graphs, making it difficult to store multiple snapshots in memory. Thus, the incremental graph representation learning problem imposes the restriction of no access to historical graph snapshots, in contrast to most dynamic graph embedding methods. Specifically, to learn the node embeddings $\{\boldsymbol{e}_v^t \in \mathbb{R}^d \ \forall v \in \mathcal{V}\}$ at time T, we require a model to only access to the snapshot $\mathcal{G}^T$ and a *summary* of the historical snapshots. For example, DynGEM (Goyal et al., 2017) is an example of an incremental embedding method that uses the embeddings learned at step $t-1$, as initialization to learn the embeddings at t.

We propose an extension of our self-attentional architecture named IncSAT to explore solving the incremental graph representation learning problem. To learn node representations at T, we first incrementally train multiple models at $1 \leq t \leq T$. Unlike the original DySAT where structural self-attention is applied at each snapshot, IncSAT applies the structural block only at the latest graph snapshot $\mathcal{G}^t$. We enable incremental learning by storing the intermediate output representations $\{\boldsymbol{h}_v^T \ \forall v \in \mathcal{V}\}$ of the structural block. As illustrated in Figure 3, these intermediate output representations of historical snapshots $(1 \leq t < T)$ can be directly loaded from previously saved results at $1 \leq t < T$. Thus, the structural information of the previous historical snapshots are *summarized* in the stored intermediate representations. The temporal self-attention is only applied to the current snapshot $\mathcal{G}^T$ over the historical representations of each node to compute the final node embeddings $\{\boldsymbol{e}_v^T \ \forall v \in \mathcal{V}\}$ at T, which are trained on random walks sampled from $\mathcal{G}^T$.

We evaluate IncSAT using the same experimental setup, with minor modifications in hyper-parameters. We use a dropout rate of 0.4 in both the structural and temporal self-attention layers. From our preliminary experiments, we find that a higher dropout rate in the structural block can facilitate avoiding over-fitting the model to the current graph snapshot. In Table 4, we report the performance of IncSAT in comparison to DySAT and DynGEM, which is the only one that can support incremental training from our baseline models. The results show that IncSAT achieves comparable performance to DySAT on most datasets while significantly outperforming DynGEM, albeit with minimal hyper-parameter tuning.

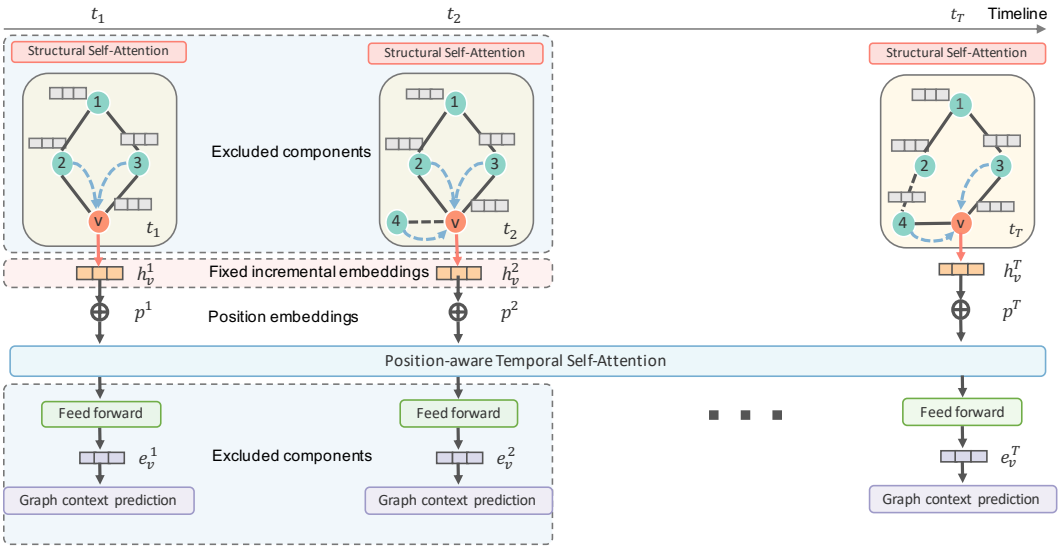

Figure 3: Neural architecture of IncSAT: the components that are excluded from DySAT are wrapped by dashed blue rectangles. The intermediate node representations are directly loaded from saved models trained previously.

C Details on Hyper-parameter Settings and Tuning

In DySAT, the objective function (Eqn. 5) utilizes positive pairs of nodes co-occurring in fixed-length random walks. We follow the strategy of Deepwalk (Perozzi et al., 2014) to sample walks 10 walks of length 40 per node, each with a context window size of 10. We use 10 negative samples per positive pair, with context distribution (P_n^t) smoothing over node degrees with a smoothing parameter of 0.75, following (Perozzi et al., 2014; Grover & Leskovec, 2016; Hamilton et al., 2017). During training, we apply L_2 regularization with $\lambda = 5 \times 10^{-4}$ and use dropout rates (Srivastava et al., 2014) of 0.1 and 0.5 in the self-attention layers of the structural and temporal blocks respectively. We use the validation set for tuning the learning rate in the range of $\{10^{-4}, 10^{-3}\}$ and negative sampling ratio w_n in the range $\{0.01, 0.1, 1\}$.

We tune the hyper-parameters of all baselines following their recommended guidelines. For node2vec, we use the default settings as in the paper, with 10 random walks of length 80 per node and context window of 10, trained for a single epoch. We tune the in-out and return hyper-parameters, p, q using grid-search, in the range $\{0.25, 0.50, 1, 2, 4\}$ and report the best results. In case of GraphSAGE, we train a two layer model with respective neighborhood sample sizes 25 and 10, for 10 epochs, as described in the original paper. We evaluate the embeddings at each epoch on the validation set, and choose the best for final evaluation. Note that the results of GraphSAGE reported in Table 2 represent that of best-performing aggregator in each dataset.

For Know-evolve, we tune the two weight-scale hyper-parameters in the range $\{10^{-4}, 10^{-3}, 0.01, 0.1\}$, learning rates in $\{10^{-4}, 10^{-3}\}$ and choose the best performing model. DynamicTriad (Zhou et al., 2018) was tuned using their two key hyper-parameters determining the effect of smoothness and triadic closure, β_0 and β_1 in the range $\{0.01, 0.1, 1, 10\}$, as advised, while using recommended settings otherwise. We use the L_1 operator ($|\boldsymbol{e}_u^t - \boldsymbol{e}_u^t|$) instead of Hadamard, as recommended in the paper, which also gives better performance. For DynGEM, we tune the different scaling and regularization hyper-parameters, $\alpha \in \{10^{-6}, 10^{-5}\}$, $\beta \in \{0.1, 1, 2, 5\}$, $\nu_1 \in \{10^{-6}, 10^{-4}\}$ and $\nu_2 \in \{10^{-6}, 10^{-4}\}$, while using other default configurations.

D Additional Dataset Details

In this section, we provide some additional, relevant dataset details. Since dynamic graphs often contain continuous timestamps, we split the data into multiple snapshots using suitable time-windows

Method	Enron		UCI		Yelp		ML-10M	
	Micro-AUC	Macro-AUC	Micro-AUC	Macro-AUC	Micro-AUC	Macro-AUC	Micro-AUC	Macro-AUC
DySAT	**85.71** $\pm$ 0.3	**86.60** $\pm$ 0.2	**81.03** $\pm$ 0.2	**85.81** $\pm$ 0.1	**70.15** $\pm$ 0.1	**69.87** $\pm$ 0.1	**90.82** $\pm$ 0.3	**93.68** $\pm$ 0.1
DynGEM	67.83 $\pm$ 0.6	69.72 $\pm$ 1.3	77.49 $\pm$ 0.3	79.82 $\pm$ 0.5	66.02 $\pm$ 0.2	65.94 $\pm$ 0.2	73.69 $\pm$ 1.2	85.96 $\pm$ 0.3
IncSAT	84.36 $\pm$ 0.2	85.43 $\pm$ 0.3	76.18 $\pm$ 0.5	85.37 $\pm$ 0.2	69.54 $\pm$ 0.1	68.73 $\pm$ 0.3	80.13 $\pm$ 0.4	91.14 $\pm$ 0.2

Table 4: Experimental results of IncSAT in comparison to DySAT (micro and macro averaged AUC with standard deviation)

Method	Enron		UCI		Yelp		ML-10M	
	Micro-AP	Macro-AP	Micro-AP	Macro-AP	Micro-AP	Macro-AP	Micro-AP	Macro-AP
node2vec	84.26 $\pm$ 0.8	84.11 $\pm$ 1.1	80.22 $\pm$ 0.4	81.12 $\pm$ 0.5	66.46 $\pm$ 0.2	63.82 $\pm$ 0.2	88.86 $\pm$ 0.2	88.71 $\pm$ 0.3
G-SAGE	83.99* $\pm$ 0.6	84.02* $\pm$ 0.6	75.91* $\pm$ 0.6	82.36* $\pm$ 0.2	58.81† $\pm$ 0.1	55.84† $\pm$ 0.2	85.45‡ $\pm$ 0.3	90.26‡ $\pm$ 0.2
G-SAGE + GAT	72.60 $\pm$ 0.5	74.75 $\pm$ 0.9	66.77 $\pm$ 0.4	76.30 $\pm$ 0.4	62.43 $\pm$ 0.1	61.49 $\pm$ 0.3	81.69 $\pm$ 0.6	82.35 $\pm$ 0.2
DynamicTriad	82.06 $\pm$ 0.9	81.22 $\pm$ 0.9	76.21 $\pm$ 0.8	80.05 $\pm$ 0.6	61.29 $\pm$ 0.3	60.79 $\pm$ 0.3	89.91 $\pm$ 0.2	89.61 $\pm$ 0.3
Know-Evolve	57.68 $\pm$ 1.2	60.71 $\pm$ 1.7	66.99 $\pm$ 0.5	77.49 $\pm$ 0.2	53.98 $\pm$ 0.2	56.44 $\pm$ 0.2	75.64 $\pm$ 0.5	79.97 $\pm$ 0.3
DynGEM	70.37 $\pm$ 0.5	72.35 $\pm$ 1.0	78.78 $\pm$ 0.3	81.71 $\pm$ 0.3	**68.02** $\pm$ 0.2	**68.09** $\pm$ 0.2	80.65 $\pm$ 0.9	89.43 $\pm$ 0.2
DySAT	**86.82** $\pm$ 0.3	**88.25** $\pm$ 0.2	**80.88** $\pm$ 0.2	**85.96** $\pm$ 0.1	65.81 $\pm$ 0.1	66.76 $\pm$ 0.1	**93.03** $\pm$ 0.2	**94.92** $\pm$ 0.1

Table 5: Experiment results on dynamic link prediction (micro and macro average precision with standard deviation). We show GraphSAGE (denoted by G-SAGE) results with the base performing aggregators for each dataset ($*$ represents GCN, † represents LSTM, and ‡ represents max-pooling).

such each snapshot has an equitable yet reasonable number of interactions (communication/ratings). In each snapshot, the weight of a link is determined by the number of interactions between the corresponding pair of users during that time-period. The pre-processed versions of all datasets will be made publicly available, along with the scripts used for processing the raw data.

Communication Networks. The original un-processed Enron dataset is available at `https://www.cs.cmu.edu/~./enron/`. We use only the email communcations that are between Enron employees, *i.e.*, sent by an Enron employee and have at least one recipient who is an Enron employee. A time-window of 2 months is used to construct 16 snapshots, where the first 5 are used as warm-up (due to sparsity) and the remaining 11 snapshots for evaluation.

The UCI dataset was downloaded from `http://networkrepository.com/opsahl_ucsocial.php`. This dataset contains private messages sent between users over a span of six months, on an online social network platform at the University of California, Irvine. The snapshots are created using their communication history with a time-window of 10 days. We discard/merge the terminal snapshots if they do not contain sufficient communications.

Rating Networks. We use the Round 11 version of the Yelp Dataset Challenge `https://www.yelp.com/dataset/challenge`. To extract a cohesive subset of user-business ratings, we first select all businesses in the state of Arizona (the state with the largest number of ratings) with a selected set of restaurant categories. Further, we filter the data to retain only users and business which have at-least 15 ratings. Finally, we use a time-window of 6 months to extract 12 snapshots in the period of 2009 to 2015.

The ML-10m dynamic user-tag interaction network was downloaded from `http://networkrepository.com/ia-movielens-user2tags-10m.php`. This dataset depicts the tagging behavior of MovieLens users, with the tags applied by a user on her rated movies. We use a time-window of 3 months to extract 13 snaphots over the course of 3 years.