Learning to Represent the Evolution of Dynamic Graphs with Recurrent Models

Aynaz Taheri
University of Illinois at Chicago
Chicago, Illinois
ataher2@uic.edu

Kevin Gimpel
Toyota Technological Institute at Chicago
Chicago, Illinois
kgimpel@ttic.edu

Tanya Berger-Wolf
University of Illinois at Chicago
Chicago, Illinois
tanyabw@uic.edu

ABSTRACT

Graph representation learning for static graphs is a well studied topic. Recently, a few studies have focused on learning temporal information in addition to the topology of a graph. Most of these studies have relied on learning to represent nodes and substructures in dynamic graphs. However, the representation learning problem for entire graphs in a dynamic context is yet to be addressed. In this paper, we propose an unsupervised representation learning architecture for dynamic graphs, designed to learn both the topological and temporal features of the graphs that evolve over time. The approach consists of a sequence-to-sequence encoder-decoder model embedded with gated graph neural networks (GGNNs) and long short-term memory networks (LSTMs). The GGNN is able to learn the topology of the graph at each time step, while LSTMs are leveraged to propagate the temporal information among the time steps. Moreover, an encoder learns the temporal dynamics of an evolving graph and a decoder reconstructs the dynamics over the same period of time using the encoded representation provided by the encoder. We demonstrate that our approach is capable of learning the representation of a dynamic graph through time by applying the embeddings to dynamic graph classification using a real world dataset of animal behaviour.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Learning latent representations**; *Unsupervised learning*.

KEYWORDS

dynamic graph; representation learning; recurrent models

ACM Reference Format:
Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. 2019. Learning to Represent the Evolution of Dynamic Graphs with Recurrent Models. In *Companion Proceedings of the 2019 World Wide Web Conference (WWW '19 Companion), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3308560.3316581

1 INTRODUCTION

Dynamic graphs are a popular model for high level representation, characterization, and analysis of real world dynamic systems. Interactions among the unit entities of a dynamic system are typically modeled and represented by a dynamic graph or a dynamic network, also referred to as a temporal graph [21, 22]. A dynamic graph is a graph or a network in which the edges between the nodes change over time, and nodes can appear and disappear from the network during the network evolution. Dynamic graphs are often defined as time-ordered sequences of network snapshots, and each network snapshot models the interactions between the nodes over a unit interval of time.

Graph representation learning approaches gained significant attention in downstream graph analysis applications in recent years. Several studies focused on learning to represent nodes and edges in dynamic graphs [16, 29, 39]. However, the ultimate goal of these approaches is still far from capturing both the topological and temporal features of entire time-varying graphs. Recently, remarkable achievements have been obtained for representation learning of whole static graphs [1, 27, 35, 40], but a comprehensive fundamental framework for processing dynamic graphs is still lacking. There are numerous differences between static and dynamic graphs that cause methods for embedding static graphs to not be applicable to dynamic graphs. The possibility of information diffusion through a series of time-varying graphs is a complex process and depends on the temporal dynamics of the graphs. Unlike static graphs, temporal dynamics, not just the topology of the connections, is the major determinant of information flow in dynamic graphs. Therefore, an ideal approach should be able to learn about the temporal dynamics in order to represent a dynamic graph.

Our goal is to demonstrate the effectiveness of our approach for dynamic graph representation learning that is able to learn the topology of the graph as well as the temporal dynamics of the graph. To that end, we propose an unsupervised approach for combining the power of static graph representation methods and the success of recurrent neural networks in learning sequences of events. The proposed model empowers the representation learning approaches to formulate the dynamic graph representation learning through gated graph neural networks (GGNNs) [25], and recurrent neural networks. Our unsupervised dynamic graph representation approach can be used not only in processing labeled data, such as in dynamic graph classification, but can be also applied in many practical applications, such as anomaly detection in temporal social networks or streaming data, as well as in exploratory analysis and scientific hypothesis generation. Our approach is constructed based on a sequence-to-sequence architecture that learns the dynamic graph evolution. GGNNs are embedded in a recurrent encoder to preserve the topology of a dynamic graph at each time step. In addition, long short-term memory networks (LSTMs) are incorporated to capture the dynamics by propagating temporal information between consecutive time steps. An autoregressive decoder is leveraged to reconstruct the history of graph evolution from the encoder hidden representation. To the best of our knowledge, this paper presents the first approach in learning to represent the entire graph in a dynamic setting. We demonstrate the efficacy of our approach in a dynamic classification task using a real dataset of baboon behaviour. We provide several baselines to demonstrate the efficacy of our approach in the classification task and present quantitative analysis to indicate the effects of considering dynamics in our representation learning method.

2 RELATED WORK

We briefly discussed some of the prior related work in section 1. In this section, we discuss the static and dynamic graph representation learning approaches in more detail.

2.1 Static Graphs

Recently, there has been a burst of methods for embedding nodes [17, 31] and subgraphs [1, 27, 40] of static graphs into low-dimensional spaces. In this paper, we only focus on the related work for representation learning of whole graphs. Graph2vec [28] is an unsupervised method inspired by document embedding models. This approach finds a representation for a graph by maximizing the likelihood of graph subtrees given the graph embedding. However, Graph2vec still does not quite capture the global information in the graph structure since it is limited to only considering subtrees as graph representatives. Taheri *et al.* [35] proposed an unsupervised architecture using sequence-to-sequence encoder-decoder models to capture the entire graph structure.

Several approaches [11, 12, 23, 25, 30, 41] have been proposed to learn graph representations for a specific supervised task, such as graph classification or regression. The representations obtained by these approaches are tailored for a supervised task and are not determined solely based on the graph structure. Niepert *et al.* [30] developed a framework (PSCN) to learn graph representations using convolutional neural networks (CNNs). Deep Graph Convolutional Neural Network (DGCNN) [41] is another model that applies CNNs for graph classification. The main difference between DGCNN and PSCN is the way they deal with the vertex-ordering problem. Lee *et al.* [23] leveraged recurrent neural networks and an attention mechanism to find informative parts of the graph for the purpose of graph classification. Message passing neural networks [12] are another group of supervised approaches that have been recently used for graph structured data, such as molecular property prediction in chemistry [11, 12, 25]. Duvenaud *et al.* [11] introduced a CNN to create "fingerprints" (vectors that encode molecule structure) for graphs derived by molecules. The information about each atom and its neighbors are fed to the neural network, and neural fingerprints are used to predict new features for the graphs. Bruna *et al.* [3] proposed spectral networks, generalizations of CNNs on low-dimensional graphs via graph Laplacians. Henaff *et al.* [18] and Defferrard *et al.* [8] extended spectral networks to high-dimensional graphs. Scarselli *et al.* [33] proposed graph neural networks (GNN) and find node representations using random walks. Li *et al.* [25] extended GNNs with gating recurrent neural networks, GGNNs, to predict sequences from graphs. Generally, GGNNs have been used in the supervised tasks in the literature. However, our architecture is an unsupervised framework in order to capture the temporal and topological aspects of a dynamic graph.

2.2 Dynamic Graphs

Lately, there have been a few studies on representation learning for dynamic graphs. The majority of these studies focused on representation learning for individual nodes within the dynamic graphs. Nguyen *et al.* [29] suggested using temporal random walks and the skip-gram model for learning node embeddings. Also, Du *et al.* [9] proposed an extension of the skip-gram model in a dynamic setting to learn the representation of new nodes and adjust the old ones. Trivedi *et al.* [36] presented an architecture, Know-Evolve, for node representation learning in a temporal knowledge graph using recurrent neural networks and temporal point processes. DyRep [37] extended Know-Evolve with a two-time scale process that captures temporal node interactions in addition to the topological evolution. Sankar *et al.* [32] proposed a framework for node representation learning including two consecutive self-attention modules for aggregation of structural and temporal information respectively. DyGEM [16] incrementally computes node representations by initializing an autoencoder from the previous step. Goyal *et al.* [15] extended DyGEM by adding recurrent neural autoencoders in order to capture more accurate temporal information of node embeddings. DynamicTriad [42] imposed triad to learn the node embeddings while preserving the temporal information. Chen *et al.* [5] modified LSTMs in order to process dynamic networks and predict the upcoming links in the future. Wang *et al.* [38] introduced a small-scale method for representation learning of a series of transition graphs in the area of driving analysis. The approach is tuned for processing small-size graphs with a fixed number of nine nodes, and the entire adjacency matrix of the graph is flattened to a vector and used as the initial representation of the graph and input of the system. However, this approach does not scale to processing large-size, sparse, and variable number of nodes through time. In contrast, our approach focuses on the entire graph representation learning and preserves both topological and temporal properties of a graph.

3 PROBLEM DEFINITION

Dynamic graph: a dynamic graph is an ordered sequence of T graph snapshots: $G = \langle G_1, G_2, \dots, G_T \rangle$. Graph $G_t = (V_t, E_t)$ models the state of a dynamic system at the interval $[t, t + \Delta t]$, for some fixed Δt. The dynamic graph G may have a subset of nodes from the set V at each time step, $V_t \subset V$. Each node $v \in V_t$ takes a unique identification value from $1, \dots, |V|$, and edge $e^t_{kj} \in E_t$ is a pair of nodes $(k, j) \in \{E_t : V_t \times V_t\}$ and represents an edge at time step t. The initialization of the embedding (node representation) for node v is denoted by $x_v \in \mathbb{R}^d$. The initialization of the node embedding is explained in Section 7. G_t is represented by an adjacency matrix $A_t \in \mathbb{R}^{n \times n}$, where n is the number of nodes and where $a^t_{kj} = 1$ if there is an edge between nodes k and j at time step t, and $a_{kj}^t = 0$ otherwise.

Timestep history: a sequence of graphs seen before time step t is called the history of G_t and represented by: $H_{G_t} = \langle G_{t-w}, G_{t-w+1}, G_{t-w+2}, \ldots, G_{t-1} \rangle$. H_{G_t} indicates a window that we have observed in the past w time steps in order to learn the graph evolution.

Graph embedding: we seek to learn a mapping function $\Phi(G_t|H_{G_t})$ that embeds a graph G_t into $\mathbb{R}^k$. The goal is to learn graph embeddings such that graphs having similar topological structure and temporal dynamics lie close to one another in the embedding space. We focus on processing undirected graphs in this work, but it is trivial to extend our models to directed graphs.

4 PRELIMINARIES AND NOTATION

Graph Neural Networks (GNNs) [14, 33] are a well-known neural network architecture for processing graph structures, and the main idea is to learn the topology of a graph via an iterative propagation procedure. These methods are sometimes referred to as Message Passing Neural Networks (MPNNs) [13] in the literature. This architecture learns node representations using a message propagation method and then aggregates the representation to find an embedding for the entire graph. The message propagation procedure runs for several steps, and updates the hidden states of the nodes at each iteration. Finally, a representation is computed for the whole graph at the last iteration based on those hidden states.
Gated Graph Neural Networks (GGNNs). Li et al. [25] proposed GGNNs as an extension of GNNs in order to learn the reachability across the nodes in a graph. They used Gated Recurrent Units (GRU) [6] to propagate a message from a node to all its reachable nodes. This allows backpropagation to unroll the recurrence in several steps and causes the hidden states of all reachable nodes to be updated. In the following we formally define the propagation model in GGNNs. Let A denote the input adjacency matrix of a graph and we will use A_v to denote row v of that matrix. Let x_v denote the initial embedding of a node v, which can be obtained in many ways and we will discuss the initialization used in this work in Section 7. We use n_i^v to indicate the hidden state of a node v at iteration i. With this notation, we have the following propagation model.

We initialize the hidden states of each node n_0^v at the beginning as follows:

$$n_0^v = x_v \tag{1}$$

Each propagation step passes information from the neighbors of a node v to learn its embedding a_i^v at propagation step i:

$$a_i^v = A_{v:}[n_{i-1}^1 \ldots n_{i-1}^{|V|}] + b \tag{2}$$

where b is a bias. Note that after several iterations of this step the information is passed among all the reachable nodes to learn the embedding of each node.

The remaining equations are the update (Equation 3) and reset (Equation 4) gates of the GRU, that update the nodes' hidden states by incorporating information from the previous iterations of the propagation step. The matrices W, U are the parameter matrices for the GRU, σ is a logistic sigmoid activation function, and tanh is a hyperbolic function to add non-linearity to produce the first estimate $\widetilde{n}_i^v$ of an embedding of a node v at step i. (The symbol $\odot$ denotes elementwise matrix multiplication.) The final embedding n_i^v of a node v at step i is given by Equation 6:

$$\begin{aligned} z_i^v &= \sigma(W^z a_i^v + U^z n_{i-1}^v) && (3)\\ r_i^v &= \sigma(W^r a_i^v + U^r n_{i-1}^v) && (4)\\ \widetilde{n}_i^v &= \tanh(W a_i^v + U(r_i^v \odot n_{i-1}^v)) && (5)\\ n_i^v &= (1 - z_i^v) \odot n_{i-1}^v + z_i^v \odot \widetilde{n}_i^v && (6) \end{aligned}$$

Long short-term memory (LSTM). We provide a brief overview of LSTMs [20]. An LSTM is a recurrent neural network (RNN) designed to model long-distance dependencies in sequential data. We denote the input vector at time t by g_t and we denote the hidden vector computed at time t by h_t. At each time step, an LSTM computes a memory cell vector c_t, an input gate vector i_t, a forget gate vector f_t, and an output gate vector o_t:

$$\begin{aligned} i_t &= \sigma(W_i g_t + U_i h_{t-1} + K_i c_{t-1} + b_i)\\ f_t &= \sigma(W_f g_t + U_f h_{t-1} + K_f c_{t-1} + b_f)\\ o_t &= \sigma(W_o g_t + U_o h_{t-1} + K_o c_{t-1} + b_o) && (7)\\ c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_c g_t + U_c h_{t-1} + b_c)\\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

Here, again, $\odot$ denotes elementwise multiplication, σ is the logistic sigmoid function, each W is a weight matrix connecting inputs to particular gates (denoted by subscripts), each U is an analogous matrix connecting hidden vectors to gates, each K is a diagonal matrix connecting cell vectors to gates, and each b is a bias.
Encoder-Decoder: In this paper, we use a sequence-to-sequence framework to train our encoder-decoder model. We refer to Equation 7 as an "LSTM encoder" because it is the main step that converts an input sequence into a sequence of hidden vectors h_t. We will also use a type of LSTM that predicts the next item $\bar{x}$ in the sequence from h_t. This architecture, which we refer to as an "LSTM decoder," adds the following:

$$\bar{g} = f(h_t) \tag{8}$$

where f is a function that takes a hidden vector and outputs a predicted observation $\bar{g}$. With symbolic data, f typically computes a distribution over symbols and returns the most probable symbols. When using continuous inputs, f could be an affine transform of h_t followed by a nonlinearity.

5 RECURRENT MODEL FOR DYNAMIC GRAPHS

Given the standard notation and existing methodology described in section 4 and the problem definition in section 3, we describe our architecture to learn the embedding of a dynamic graph $\Phi(G_t|H_{G_t})$ in this section. The proposed architecture is composed of a recurrent encoder that projects the timestep history H_{G_t} into a k dimensional space. GGNNs are applied to model the graph topology at each time step, and more specifically provide the input for the encoder. The recurrent structure of the encoder allows us to embed the G_t using the hidden representation of the timestep history H_{G_t} and the graph topology at time step t.
DyGGNN: The model includes three main components: 1) a *GGNN* to capture the topology of the graph at time step t, 2) an *encoder*

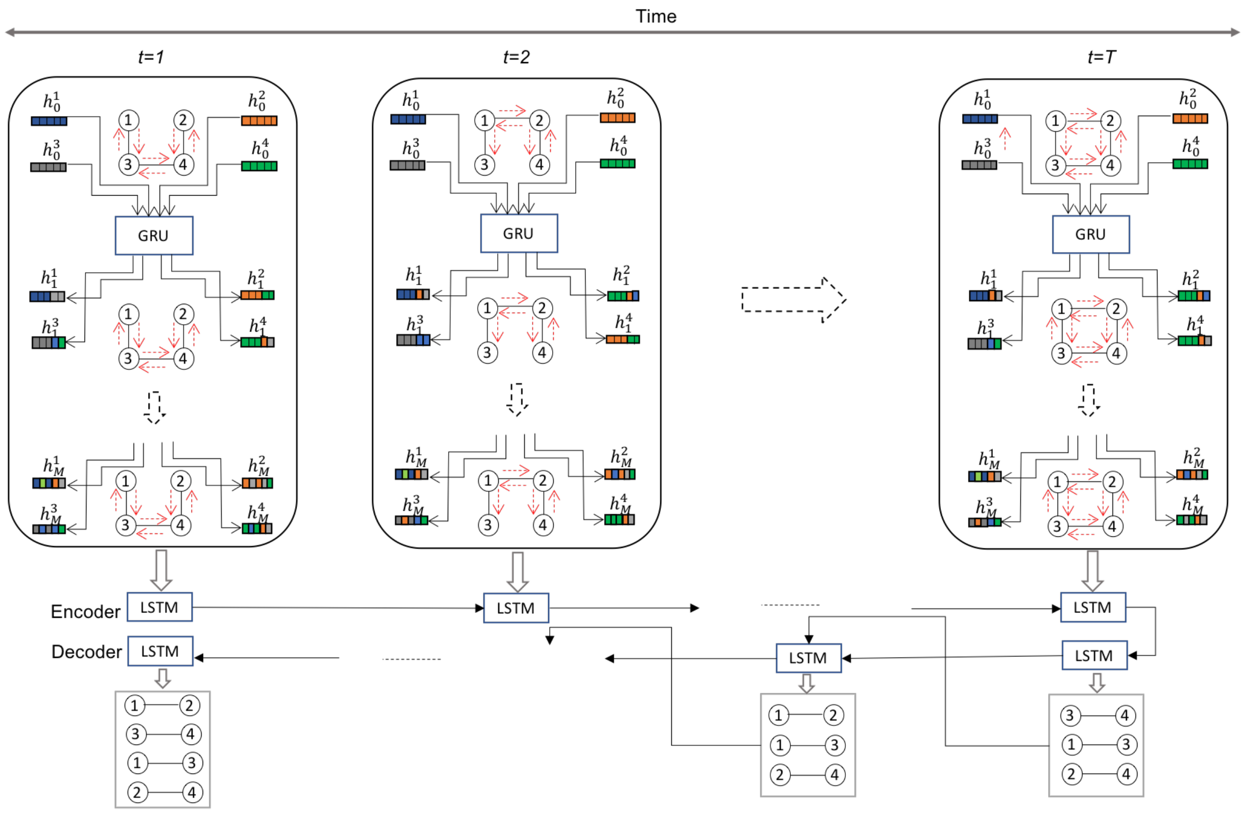

Figure 1: DyGGNN model schematic. A GGNN models the graph topology at each time step and provides the input to the LSTM encoder. The hidden representation of a sequence of graphs is used by the decoder to reconstruct the dynamic graph evolution.

that projects the graph evolution over a window of T time steps into a k dimensional space, and 3) a *decoder* which reconstructs the structure of the dynamic graph at each time step using its hidden representation. Figure 1 shows the architecture of our approach for representation learning of a dynamic graph over T time steps. In the following, we explain the components of this architecture in detail.

The GGNN builds a graph representation for G_t by considering its topological structure at time step t. After M steps of message propagation in G_t, we use an average pooling of the nodes' hidden states (Equation 6) as the representation of the entire G_t, and we refer to this as a static embedding of the graph:

$$Emb_{st}(G_t) = Avg(h_M^v)$$

Given the static embedding of the graph $Emb_{st}(G_t)$ and its history H_{G_t}, we use an LSTM encoder to project G_t into a hidden representation that takes the graph dynamics into account. The LSTM encoder, $LSTM_{enc}$, passes the information of the dynamic graph G over an observation window of size w and computes the dynamic embedding G_t using knowledge about the graph topology from the past w time steps.

$$h_t^{enc} = LSTM_{enc}(Emb_{st}(G_t), h_{t-1}^{enc}) \tag{9}$$

We refer to h_t^{enc} as the dynamic embedding of the graph, $Emb_{dy}(G_t)$. The main goal of the decoder, $LSTM_{dec}$, is to reconstruct the history of the graph H_{G_t} in the observed window with size w. The decoder is as an autoregressive model and predicts the topology of the graph at time step t given the recently predicted graphs at previous time steps. The decoder uses h_T^{enc} to initialize its first hidden state. The set of edges of the graph is the prediction of the decoder at each time step. The f function in Equation 8 is instantiated as follows:

$$h_t^{dec} = LSTM_{dec}(\overline{Edges}(G_{t-1}), h_{t-1}^{dec}) \tag{10}$$

We use a sigmoid transformation σ to predict the existence/nonexistence of an edge at time step $t-1$ (here W are the learned parameters and b is a bias):

$$\overline{Edges}(G_{t-1}) = \sigma(h_{t-1}^{dec} * W + b) \tag{11}$$

6 TRAINING

Let O be a set of temporal windows where each window $o \in O$ has a length of w. The parameter space of the models includes parameters related to $GGNN$, $LSTM_{enc}$, $LSTM_{dec}$, $W \in \mathbb{R}^{|h^{dec}| \times |V|^2}$ and $b \in \mathbb{R}^{|V|^2}$. We use a cross-entropy loss function to train our model:

$$Loss_{CE} = \sum_{o \in O} \sum_{t=1}^{|o|} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} A_{ij}^{t} \log(\overline{Edges}(G_t)_{ij})) \qquad (12)$$

In order to train an efficient model, we need to pay attention to the choice of the temporal window size, w, since we define the connectedness of the temporal graph over the length of the window. There may be some very long temporal paths that cannot be captured with a window size of w. On the other hand, we cannot consider very long windows in practice, because we face the gradient vanishing problem during backpropagation. Some approaches [10, 19] truncate very long sequences of individual events to a batch of shorter independent sequences and use backpropagation though time over the mini-batches of those sequences. However, in our task we have a dynamic graph G which is a sequence of graph snapshots, and we need to take care of the sequence of nodes' hidden states. We cannot consider the sequences of nodes independently because we need to consider both temporal and topological structure of the graph. Trivedi et al. [36] suggested using a different type of mini-batches for event detection on edges of a temporal graph. They select k sequential events through time and made a batch with size k from the events. Therefore, at each mini-batch, only parameters related to nodes and edges constructing those k events are updated. Their approach cannot fully utilize the history of nodes, topology, and dynamics of a temporal graph. Using a sliding observation window, we decompose the dynamic graph into several dynamic graphs. The sequential windows are grouped together to make a mini-batch (Algorithm 1). In this way, we preserve not only the temporal dependency at each window, but also we capture the implicit relations between sequential windows, since the model parameters are updated considering the whole mini-batch input data.

Algorithm 1 Mini-batch Backpropagation on Temporal Graphs

```
Input: Dynamic graph $G : (G_1, G_2, \ldots, G_T)$, Window size $w$,
Batch size $b$, Sliding gap $s$
$cur_{win} = 0, b_{ind} = 0, minbatch = \{\}$
while $cur_{win} < T$ do
  while $b_{ind} < b$ do
    $sample = G[cur_{win} : cur_{win} + w]$
    $minibatch.Add(sample)$
    $b_{ind} = b_{ind} + 1$
    $cur_{win} = cur_{win} + s$
  end while
  Use $minibatch$ to find the $\overline{Edges}$ (Eq. 11)
  Minimize loss function (Eq. 12) through $w$ time steps
  $minibatch = \{\}, b_{ind} = 0$
end while
```

7 EXPERIMENTS

In this section, we evaluate our dynamic representation learning approach using our proposed model. We use our learned representation for a dynamic graph classification task. At each time step t, the graph G_t has a label l from the set L. Our unsupervised learned representations for G_t are used in the classification task.

7.1 Dataset

We use dynamic graphs of a troop of GPS-tracked baboons living in the wild in Mpala Research Centre, Kenya [7, 34]. The adult and sub-adult members of the troop were fitted with GPS collars, collecting data points at 1Hz for 12 hours (6am to 6pm) for 28 days. The data are a 2-day subset of the dataset from 16 adult and sub-adult members of the troop. The subset was chosen due to the availability of the activity labels, provided by biologists and extended by sequence classification approaches [24]. Amornbunchornvej *et al.* [2] constructed dynamic graphs of individuals following the behavior patterns of each other and performed the activity classification task. The dynamic graph of the first day has 23259 time steps and the second day has 19098 time steps. There are four group level activities in the baboon dataset: sleeping, hanging-out, coordinated non-progression, and coordinated progression. One of the four labels is assigned to the dynamic graph at each time step by the domain experts. We report two results, following the configuration baselines in [2, 24]: using the labeled first day of data to classify the second day's activities (Baboon (day 1)); and using the second day's labeled data to classify the first day's activities (Baboon (day 2)).

7.2 Baselines

We compare our results with several baselines:

- Li *et al.* [24] proposed an Adversarial Sequence Tagging (AST) to perform activity labeling classification for the baboon dataset.
- Amornbunchornvej *et al.* [2] used dynamic graph features at each time step to predict the activity label using a linear discriminant analysis (LDA).
- We also compare with other baselines that learn static graph representations. Taheri *et al.* [35] proposed an unsupervised sequence-to-sequence (S2S) approach for static graph representation learning. We use this approach to find the representation of G_t without considering its history.
- Moreover, we use a supervised GGNN classifier as a baseline to classify G_t based on the graph topology at the same time step.

7.3 Hyperparameters

The GGNN uses four iterations of message passing at each time step. The dimensionality of the GRU and LSTMs is fixed to 100. The Adam optimizer is used for minibatch training. We set the learning rate to be 0.001 and dropout rate to 0.5. A unique identifier is assigned to each baboon in the troop. We represent identifiers via one-hot vectors to initialize the x_v in GGNNs. Figure 2 indicates the effect of window size and batch size on the classification performance. It shows that the best accuracy is obtained by larger window sizes and batch sizes, which points to the fact that longer-term dynamics matter for this task. Having larger window sizes causes an explicit larger context for representation learning during backpropagation, and having larger batch sizes causes an implicit larger context through time. Smaller window sizes are more sensitive to the batch

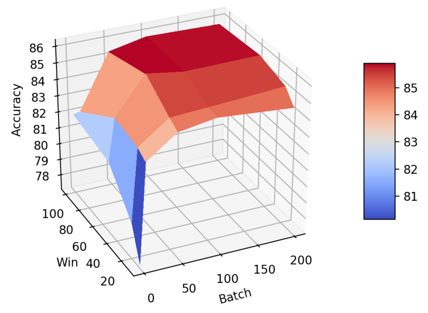

Figure 2: The effect of window size and batch size on accuracy.

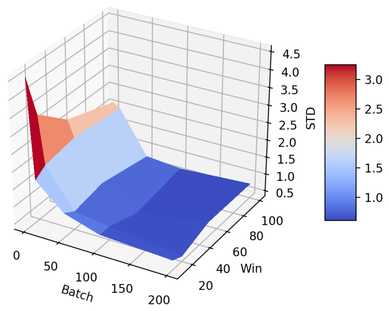

Figure 3: The effect of window size and batch size on standard deviation.

size, and small batch sizes do not perform well especially with small window sizes. Figure 2 shows that we get the best results with window sizes larger than 50 and batch size larger than 50. We fixed the batch size and window size to 50 in the rest of our experiments. The standard deviations of different configuration of window sizes and batch sizes are shown in Figure 3. Each point in the plot shows the standard deviation of 10 experiments. Larger batch sizes have an important role in the robustness of our experiments. Increasing the batch size provides smaller standard deviations while smaller window sizes are prone to greater standard deviations when the batch size is small.

7.4 Results

We use a C-SVM classifier from LIBSVM [4] with a radial basis kernel for multi-class classification. The data of one day is used for training and tuning the regularization and kernel hyperparameters of the SVM via cross-validation and the other day is used for the test. Each experiment is repeated 10 times and we report average accuracies and standard deviations. We compare our proposed models with other baselines in Table 1. we exceed all other baselines and the performance shows that dynamics matter and incorporating temporal information in embedding methods improves graph representations.

Table 1: Activities classification prediction accuracy in first two days of baboon data

Method	Baboon Day 1	Baboon Day 2
AST	77.30	69.22
LDA	87.20	70.82
GGNN	84.58±1.2	81.38±1.1
S2S	84.66±1.3	83.87±1.5
DyGGNN	88.02±1.8	86.21±1.7

7.5 Model analysis

Our model computes representations for G_t from two points of view: A static representation of the topology of the graph at time step t: $Emb_{st}(G_t)$ and a dynamic representation of the graph: $Emb_{dy}(G_t)$. Figure 4 shows the difference between the representations found by the $Emb_{st}(G_t)$ and $Emb_{dy}(G_t)$ functions. The accuracy at time step t in the plot shows the accuracy of the model in the interval $[0 \ldots t]$. The generated representations from $Emb_{dy}(G_t)$, which include the dynamic notion of graph evolution, outperform the embeddings obtained by the $Emb_{st}(G_t)$, which consider only the topology of the graph at time step t. Furthermore, Figure 4 highlights that training the model through time improves the performance of the dynamic embedding significantly in comparison with a static representation.

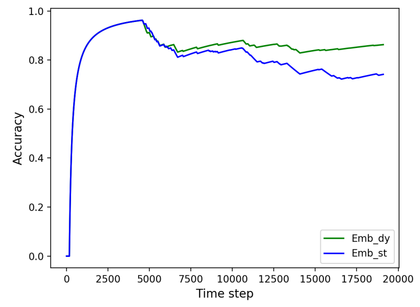

Figure 4: The classification accuracy of the model in the interval of $[0 \ldots t]$ for the two embeddings of $Emb_{st}(G_t)$ and $Emb_{dy}(G_t)$.

Figure 5 shows the representation of the baboon dataset using t-SNE [26]. Each point shows the graph assigned to one of the time steps. The four class labels were not used when learning the graph representation, but we still find the graphs are well clustered. Since these are human observed activities and there is a natural transition in behavior (these are not discrete behaviors), there is a continuity of labels in the representations. We actually do not expect *a priori* cluster separation. Note, that most of the activity is “hanging out", which is biologically consistent. The sleeping activity is seen at the bottom center of the right figure in Figure 5 and can be thought of as a special kind of “hanging out”. The “coordinated

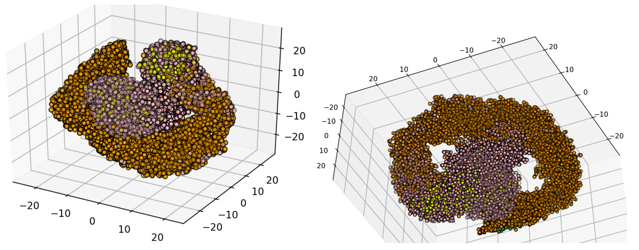
Figure 5: Visualization of the baboon dataset in a 3-dimensional space (Orange: Hanging-out, Pink: Coordinated progression, Yellow: Coordinated non-progression, Green: Sleeping)

non-progression" is a transitional activity from "hanging-out" to "coordinated progression".

8 CONCLUSIONS

We propose an efficient approach for representation learning of entire graphs in a dynamic setting. We evaluate our approach in a real-world dataset of animal behaviour and show that our model achieves significantly better performance in comparison with state-of-the-art models that learn the graph representation in a static setting. The promising results confirm our approach as a solid foundation for dynamic graph analysis.

REFERENCES

[1] B. Adhikari, Y. Zhang, N. Ramakrishnan, and B. A. Prakash. 2017. Distributed Representations of Subgraphs. In *DaMNet*.

[2] Chainarong Amornbunchornvej, Ivan Brugere, Ariana Strandburg-Peshkin, Damien R. Farine, Margaret C. Crofoot, and Tanya Y. Berger-Wolf. 2018. Coordination Event Detection and Initiator Identification in Time Series Data. *ACM Trans. Knowl. Discov. Data* 12, 5 (2018), 53:1–53:33.

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral Networks and Locally Connected Networks on Graphs. *CoRR* (2013).

[4] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: a library for support vector machines. *ACM TIST* 2 (2011).

[5] Jinyin Chen, Xuanheng Xu, Yangyang Wu, and Haibin Zheng. 2018. GC-LSTM: Graph Convolution Embedded LSTM for Dynamic Link Prediction. *arXiv preprint arXiv:1812.04206* (2018).

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[7] MC Crofoot, RW Kays, and Wikelski M. 2015. Data from: Shared decision-making drives collective movement in wild baboons. *Movebank Data Repository* (2015).

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv* (2016).

[9] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. 2018. Dynamic Network Embedding: An Extended Approach for Skip-gram based Network Embedding.. In *IJCAI*.

[10] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. 2016. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1555–1564.

[11] David Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*.

[12] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. *CoRR* (2017).

[13] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212* (2017).

[14] Marco Gori, Gabriele Monfardini, and Franco Scarselli. [n. d.]. A new model for learning in graph domains. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN'05)*, Vol. 2. IEEE, 729–734.

[15] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2018. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *arXiv preprint arXiv:1809.02657* (2018).

[16] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2017. DynGEM: Deep Embedding Method for Dynamic Graphs. *CoRR* abs/1805.11273 (2017).

[17] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*.

[18] M. Henaff, J. Bruna, and Y. LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv* (2015).

[19] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *ICLR*.

[20] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural computation* (1997).

[21] Petter Holme and Jari Saramäki. 2012. Temporal networks. *Physics reports* 519, 3 (2012), 97–125.

[22] David Kempe, Jon Kleinberg, and Amit Kumar. 2002. Connectivity and inference problems for temporal networks. *J. Comput. System Sci.* 64, 4 (2002), 820–842.

[23] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1666–1674.

[24] Jia Li, Kaiser Asif, Hong Wang, Brian D Ziebart, and Tanya Y Berger-Wolf. 2016. Adversarial Sequence Tagging.. In *IJCAI*.

[25] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated graph sequence neural networks. In *ICLR*.

[26] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *JMLR* (2008).

[27] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. 2016. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *MLG* (2016).

[28] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning Distributed Representations of Graphs. In *MLG*.

[29] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *International Workshop on Learning Representations for Big Networks at The Web Conference*.

[30] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *ICML*.

[31] B. Perozzi, R. Al-Rfou, and S Skiena. 2014. DeepWalk: Online learning of social representations. In *KDD*.

[32] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2018. Dynamic Graph Representation Learning via Self-Attention Networks. *arXiv preprint arXiv:1812.09430* (2018).

[33] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.

[34] Ariana Strandburg-Peshkin, Damien R Farine, Iain D Couzin, and Margaret C Crofoot. 2015. Shared decision-making drives collective movement in wild baboons. *Science* 348, 6241 (2015), 1358–1361.

[35] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. 2018. Learning Graph Representations with Recurrent Neural Network Autoencoders. In *KDD Deep Learning Day*.

[36] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. 2017. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *ICML*.

[37] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *International Conference on Learning Representations*.

[38] Pengyang Wang, Yanjie Fu, Jiawei Zhang, Pengfei Wang, Yu Zheng, and Charu Aggarwal. 2018. You are how you drive: Peer and temporal-aware representation learning for driving behavior analysis. In *KDD*. ACM.

[39] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2724–2743.

[40] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *KDD*.

[41] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. In *AAAI*.

[42] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process.. In *AAAI*.