Temporal Neighbourhood Aggregation: Predicting Future Links in Temporal Graphs via Recurrent Variational Graph Convolutions

Stephen Bonner [†], Amir Atapour-Abarghouei [‡], Philip T Jackson [†], John Brennan [†], Ibad Kureshi [¶], Georgios Theodoropoulos [§], Andrew Stephen McGough [‡] and Boguslaw Obara [†]
[†]Department of Computer Science, Durham University, Durham, UK,
{s.a.r.bonner, p.t.g.jackson, j.d.brennan, boguslaw.obara}@durham.ac.uk
[‡]School of Computing, Newcastle University, Newcastle, UK, {amir.atapour-abarghouei, stephen.mcgough}@newcastle.ac.uk
[§]School of Computer Science and Engineering, SUSTech, Shenzhen, China, georgios@sustec.edu.cn
[¶]Inlecom Systems, Brussels, Belgium, ibad.kureshi@inlecomsystems.com

***Abstract*—Graphs have become a crucial way to represent large, complex and often temporal datasets across a wide range of scientific disciplines. However, when graphs are used as input to machine learning models, this rich temporal information is frequently disregarded during the learning process, resulting in suboptimal performance on certain temporal inference tasks. To combat this, we introduce Temporal Neighbourhood Aggregation (TNA), a novel vertex representation model architecture designed to capture both topological and temporal information to directly predict future graph states. Our model exploits hierarchical recurrence at different depths within the graph to enable exploration of changes in temporal neighbourhoods, whilst requiring no additional features or labels to be present. The final vertex representations are created using variational sampling and are optimised to directly predict the next graph in the sequence. Our claims are supported by experimental evaluation on both real and synthetic benchmark datasets, where our approach demonstrates superior performance compared to competing methods, outperforming them at predicting new temporal edges by as much as 23% on real-world datasets, whilst also requiring fewer overall model parameters.**

***Index Terms*—representation learning, dynamic link prediction**

I. Introduction

Using graphs to represent relationships in large, complex and high-dimensional datasets has become a universal phenomenon across many scientific fields. Encompassing not only computer scientists, interested in social and citation networks [1], but biologists, studying protein interaction graphs for associations with diseases [2], chemists, who model molecule properties by treating them as graphs [3], and physicists, who use graphs to model a physical environment [4].

Using graph-based approaches enables complex data analysis, with one of the most universal being the identification of missing links within the graph, which can provide invaluable insight in many real-world scenarios. For example, the recommendation of acquaintances on social networks, new research papers to read or even new links between molecules. However, to date, almost all of the prediction work performed on graphs has been focused on analysis in solely the topological domain, ignoring the rich temporal information inherent in so much of the data represented by graphs (as seen Figure 1).

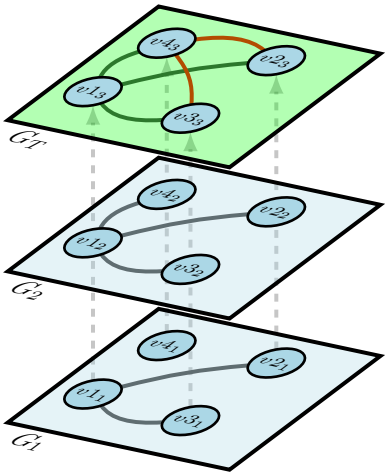


Fig. 1: The temporal link prediction task is to predict the new edges (red) in the final graph snapshot G_T (green plane) given the previous graphs G_1 and G_2.

We formally define a graph $G = (V, E)$ as a finite set of vertices V, with a corresponding set of edges E. Elements of E are unordered tuples $\{i, j\}$ where $i, j \in V$. Elements in V and E may have labels or certain associated features, although these are not required for this work. In order to perform analysis on graphs, we need a mechanism which converts the formal graph representation into a format which is amenable for machine learning – graph representation learning.

The field of graph representation learning has received significant attention as a means of analysing large, complex graphs via the use of machine learning. Graph representation learning, comprises a set of techniques that learn latent representations of a graph, which can then be used as the input

978-1-7281-0858-2/19/$31.00

to machine learning models for downstream prediction tasks [5]. The majority of graph representation learning techniques have focused upon learning vertex embeddings [6] and reconstructing missing edges [5]. As such, the goal of graph representation learning is to learn some function $f : V \rightarrow \mathbb{R}^d$ which maps from the set of vertices V to a set of embeddings of the vertices, where d is the required dimensionality. This results in f being a mapping from G to a representation matrix of dimensions $|V| \times d$, i.e. an embedding of size d for each vertex in the graph. However, the majority of graph representation learning approaches to date ignore the temporal aspect of dynamic graphs, resulting in models which perform poorly at predicting future change in a graph.

This paper introduces a new model, entitled Temporal Neighbourhood Aggregation (TNA), designed to learn vertex representations which capture both topological and temporal change by exploiting the rich information found in large dynamic graphs. To achieve this, we propose a novel model architecture combining graph convolutions with recurrent connections on the resulting vertex level representations to allow for powerful, hierarchical learning at multiple hops of a vertices neighbourhoods. This approach means the model can explore at which neighbourhood depth the most useful temporal information can be learned. Further, we aggregate the temporal neighbourhood using tools from variational inference, resulting in a more robust and stable final representation for each vertex. Our TNA model is trained end to end on temporal graphs represented as time snapshots, where the objective is to directly and accurately predict the next graph in the sequence using the embeddings alone. This results in a model, which unlike many competing approaches, requires no explicitly parameterized decoder model. In summary, our primary contributions are as follows:

- *Temporal Neighbourhood Aggregration* - Our proposed model is capable of independently learning the temporal evolutionary patterns within the neighbourhood of a vertex at different depths, resulting in superior performance at predicting future links. Moreover, our approach requires no additional vertex features, labels or random walk procedures as part of its process.
- *Variational Sampling* - More robust temporal representations and consequently accurate prediction of the next graph in the evolving sequence is made possible by our approach by sampling vertex embeddings using the principals of variational inference.
- *Model Efficacy and Scalability* - Our model contains significantly fewer parameters than competing approaches, as it does not require a parameterized decoder portion. This leads to our model being scalable to larger graphs as a result of its memory efficiency.

Our work is supported by extensive experimentation on public benchmark datasets. Further, to aid reproducibility, we open-source all of our PyTorch [7] based source-code[1] and experimentation scripts.

[1]https://github.com/sbonner0/temporal-neighbourhood-aggregation

II. Related Works

We highlight prior work in the areas of graph representation learning and temporal embeddings.

A. Graph Representation Learning

Historically, low dimensional graph representations were created via matrix factorization techniques. Examples of such approaches include Laplican eigenmaps [8] and Graph Factorization [9]. More recent models, originally used for Natural Language Processing (NLP) tasks, have been adapted to learn graph embeddings. These approaches exploit random walks to create 'sentences' which can be used as input to language-inspired models such as DeepWalk [10] and Node2Vec [5].

Graph-specific neural network based models have been created, inspired by Convolutional Neural Networks (CNN). Such approaches attempt to create a differential model for learning directly from graph structures. Many Graph CNN approaches operate in the spectral domain of the graph, using eigenvectors derived from the Laplacian matrix of a graph [1]. The Graph Convolutional Network (GCN) approach has proven to be particularly effective [1]. GCN uses a layer-wise propagation rule to aggregate information from the 1-hop neighbourhood of a vertex to create its representation. This layer-wise rule can be stacked k times to aggregate information from k hops away.

The approaches discussed thus far have been supervised, mandating the use of labels. However, graph embedding approaches exist which are based on auto-encoders - a type of neural network trained to reconstruct the input data after initially being projected into a lower dimension [11]. For example, GCNs have been used as the basis of a convolutional auto-encoder model [12], demonstrating state-of-the-art results for static link prediction.

B. Temporal Graph Embeddings

We argue that the existing approaches for temporal graph embeddings can be split in two categories: Temporal Walk and Adjacency Matrix Factorisation.

1) Temporal Walk Approaches: In an approach entitled STWalk [13], the authors aim to learn *node trajectories* via the use of random walks which learn representations that consider all the previous time-steps of a temporal graph. In the best performing approach presented, the authors learn two representations for a given vertex simultaneously which are concatenated to create the final temporal embedding. However, the approach is not end to end and requires the user to manually chose how many time steps to consider.

Yu et al. [14], propose NetWalk, which enables anomaly detection in streaming graphs via a vertex-level dynamic graph embedding model. In the approach, a collection of short random walks captured from the graph is passed into an auto-encoder based model to create the vertex representations.

Nguyen et al. [15], propose a model to incorporate temporal information when creating graph embeddings via random walks by capturing individual temporal changes within a graph. They propose a temporal random walk to create the

input data, with the approach producing more complex and rich temporal walks via a biasing process.

2) Adjacency Matrix Factorisation Approaches: Goyal et al. [16], propose a model for creating dynamic graph embeddings, entitled DynGEM. In this approach, they extend the auto-encoder graph embedding model of Structural Deep Network Embedding (SDNE) [17] to consider dynamic graphs, by using a method similar to Net2net [18], which is designed to transfer knowledge from one neural network to a second.

In a family of approaches entitled Dyngraph2vec*, comprised of DynAE, DynRNN and DynAERNN, Goyal et al. [19] further extend an SDNE type approach to incorporate temporal information in a variety of ways. The best performing of approaches, DynAERNN, uses a combination of SDNE-like dense auto-encoders, with stacked recurrent layers to learn temporal information when creating vertex embeddings. However, they do not make use of graph convolutions and require a complex decoder model to predict the next graph.

There have been attempts to incorporate temporal aspects into GCNs. However, some [20], [21] focus upon supervised learning, do not explicitly use the models to predict the future graph state and only have a single layer of recurrent connections. More recent approaches, such as GCN-GAN [22] and GC-LSTM [23] require large and complex decoder models, meaning they cannot scale to graphs of one-thousand vertices or more on current hardware, whilst also lacking the variational sampling of our approach. In comparison, EvolveGCN [24] uses recurrent layers to directly evolve the parameters of standard GCN layers which means it does not track vertex neighbourhood evolution explicitly.

One of the application areas most frequently learning temporal models on graphs is that of traffic modelling. Where approaches like [25] and [26] combine graph learning with temporal models to predict traffic movement. However, unlike these approaches we focus on creating vertex level embeddings directly optimised to predict future edges and learn change at different hops of a vertices neighbourhood.

III. Methodology

We briefly outline the proposed approach, relevant background, network architecture and the training procedure. Throughout, we make use of the notation in Table I.

A. Motivation

Many of the phenomena that are commonly represented via graph structures are known to evolve over time – Links between entities form and break in a constantly evolving stream of changes. We thus view graphs as a series of snapshots, with each graph snapshot containing the connections present at that particular moment in time. More formally, we can redefine a graph G to be a temporal graph $G' = \{G_1, G_2, ..., G_T\}$, where each graph snapshot $G_t \ \forall t \in [1, T]$ contains a corresponding vertex set V_t and edge set E_t.

A common and vital task within the field of graph mining is that of future link prediction, where the goal is to accurately predict which vertices within a graph will form a connection in the future [16]. Figure 1 highlights this future link prediction task, where the goal is to predict the new edges, coloured in red, formed in G_T, given the previous graphs in the temporal history G_1 and G_2. Any model designed to accomplish this task must learn the evolution patterns present in edge formation, even though the number of edges changing at each time point is often a small fraction of the total number.

Symbol	Definition
G	A graph with an associated set of vertices V and corresponding set of edges E.
A	The adjacency matrix of graph G, a symmetric matrix of size $\|V\| \times \|V\|$, where $(a_{i,j})$ is 1 if an edge is present and 0 otherwise.
$\hat{A}$	A normalised by its degree matrix D and its identity matrix I such that $\hat{A} = (D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}})$ [1].
X	A matrix of features for each $v \in V$, set to the identity I of A for this work.
H	The intermediate vertex representations in GCN and TNA layers.
Z	The final variationally sampled representation matrix for each $v \in V$.
G'	A temporal graph comprised of snapshots $\{G_1, G_2, ..., G_T\}$.
T	The number of snapshots in G'.
G_t	A graph from G'.
σ_s	The sigmoid activation function.
σ_r	The rectified linear activation function (ReLU).
σ_{lr}	The leaky ReLU activation function.
l	A certain layer in the model.
$W_g^{(l)}$	A weight matrix at layer l used in the GCN.
$W_s^{(l)}$	A weight matrix at layer l used in the skip connection.
$W_{\{r,u,h\}}^{(l)}$	Hidden transform matrices in the GRU.
$U_{\{r,u,h\}}^{(l)}$	Input transform matrices in the GRU.
$\mathcal{N}(\mu, \sigma)$	A multi-dimensional Gaussian distribution parametrised by vectors μ and σ.
Θ	A trainable model containing a set of parameters.

TABLE I: Definitions and Notations

We propose to tackle this by creating temporally-aware graph embeddings, which are explicitly trained to recreate a future time step of the graph. We entitle our approach Temporal Neighbourhood Aggregation (TNA), since to create a better and more meaningful representation for a certain vertex, the model is able to aggregate information about how its neighbourhood has changed in the past to more accurately predict how it will change into the future. More concretely, a temporal graph G' is input to our TNA model $\Theta(G')$ which learns a representation for each vertex in $G_t \in G'$ such that its output can accurately predict the graph G_{t+1}. Ideally, we want to create a model $\Theta()$ which can perform this temporal learning using just the sequence of graphs until G_t, such that $G_{t+1} = \Theta(G_1, ..., G_t)$. TNA is able to accomplish this, requiring no pre-processing steps which could affect the models performance (e.g. random walk procedures), no pre-computed vertex features and no additional labels.

B. Background Technologies

We first review the background technologies we are employing to make it possible, namely Graph Convolutions [1] and Recurrent Neural Networks [27], [28].

1) Graph Convolutions: To perform the graph encoding required to create the initial vertex representations, we utilise the spectral Graph Convolution Networks (GCN) [1]. One can consider a GCN to be a differentiable function for aggregating information from the immediate neighbourhood of vertices [29], [30]. A GCN takes the normalised adjacency matrix $\hat{A}$ representing a graph G, and a matrix of initial vertex level features X, and computes a new matrix of vertex level features $H = GCN(\hat{A}, X)$. X can be initialized with pre-computed vertex features, but it is sufficient to initialize with one-hot feature vectors (in which case X is the identity matrix I). A GCN can contain many layers which aggregate the data, where the operation performed at each layer by the GCN [1] is:

$$GCN^{(l)}(H^{(l)}, \hat{A}) = \sigma_r(\hat{A}H^{(l-1)}W_g^{(l)}), \tag{1}$$

where l is the number of the current layer, $W_g^{(l)}$ denotes the weight matrix of that layer, $H^{(l-1)}$ refers to the features computed at the previous layer or is equal to X at $l = 0$.

One can consider the GCN function to be aggregating a weighted average of the neighbourhood features for each vertex in the graph. Stacking multiple GCN layers has the effect of increasing the number of hops from which a vertex-level representation can aggregate information – a three layer GCN will aggregate information from three-hops within the graph to create each representation.

The original method requires GCN based models to be trained in a supervised learning framework, where the final vertex representation is tuned via labels provided for a specific task – classification being common [1], [30]. Extensions to the GCN framework have been made which allow for convolutional auto-encoders for graph datasets [12].

2) Recurrent Neural Networks (RNN): RNN are neural networks with circular dependencies between neurons. Activations of a recurrent layer are dependent on their own previous activations from a previous forward pass, and therefore form a type of internal state that can store information across time steps. They are frequently used in sequence processing tasks where the response at one time step should depend in some way on previous observations. Long Short-Term Memory (LSTM) [27] and Gated Recurrent Units (GRU) [28] are RNNs with learned gating mechanisms, which mitigate the vanishing gradient problem when back-propagating errors over a sequence of inputs, allowing the model to learn longer-term dependencies. For this work, we employ the GRU cell, as it empirically offers similar performance to an LSTM, but with fewer overall parameters. The GRU computes the output h_t, for the input vector x_t at time t in the following manner [28]:

$$\begin{aligned} u_t &= \sigma_s\big(x_t U_u^{(l)} + h_{t-1} W_u^{(l)}\big) \\ r_t &= \sigma_s\big(x_t U_r^{(l)} + h_{t-1} W_r^{(l)}\big) \\ \tilde{h}_t &= \tanh\big(x_t U_h^{(l)} + (r_t * h_{t-1}) W_h^{(l)}\big) \\ h_t &= (1 - u_t) * h_{t-1} + u_t * \tilde{h}_t, \end{aligned} \tag{2}$$

where r and u are the rest and update gates and σ_s and $\tanh$ are the sigmoid and hyperbolic tangent activation functions.

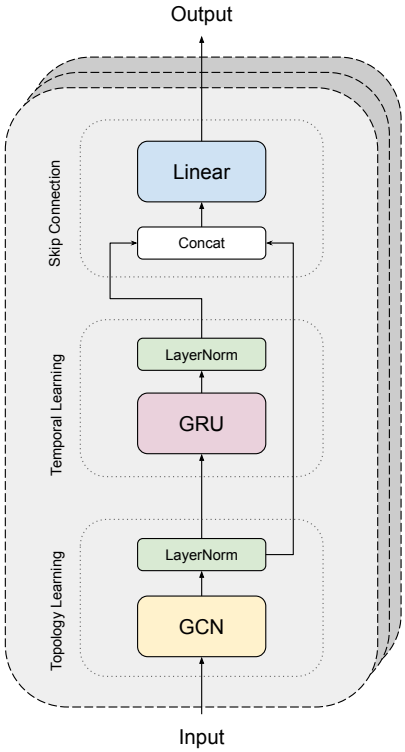


Fig. 2: An overview of the Temporal Neighbourhood Aggregation (TNA) block, which comprises a Graph Convolutional Network (GCN) layer with a Gated Recurrent Unit (GRU). The combination of the topological and temporal learning is controlled via the final linear layer.

C. Model Overview

We first detail the Temporal Neighbourhood Aggregation blocks which form the primary learning component, before describing the overall model topology and objective function.

1) TNA Block: One of the primary components of our model is the TNA block for topological and temporal learning from graphs. The overall structure of the block is illustrated in Figure 2. It is important to note that all the parameters in the block are shared through time. This allows complex temporal patterns to be learned, as well as allowing for a large reduction in the total number of parameters required by the model. Assuming that the TNA block is the first layer in the model, the flow for vertex $v \in V_t$ can be described as follows:

- The input is passed through the GCN layer, as detailed in Equation 1, which will learn to aggregate information for v from its one-hop neighbourhood to create its representation at this point in the block - h_t^{GCN}. This is then normalised using Layer Norm [31], which will ensure that the representation for each vertex is of a similar scale, this has been shown to improve the training stability and convergence rate of deep models [31].
- This normalised representation is then passed into a GRU cell a row at a time, as detailed in Equation 2, where the output of the cell will be a function of the current input as well as all the previous inputs. Meaning that the cell can learn how much of the previous neighbourhood rep-

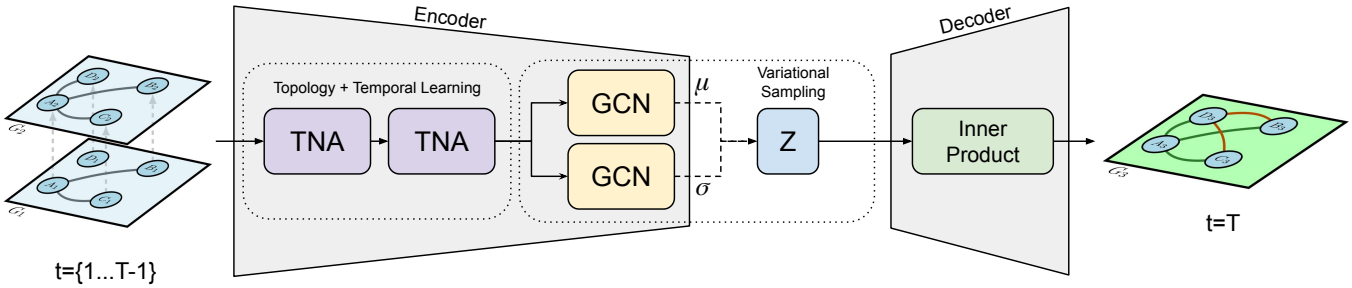


Fig. 3: The overall Temporal Neighbourhood Aggregation Model: two stacked TNA blocks learning both topological and temporal information from the first and second hop neighbourhoods of a vertex. An embedding z_t is sampled for each vertex $v_t \in V_t$ using variational inference. The inner product is then used to directly predict the next graph in the sequence.

resentation to use when creating the new representation for a given vertex h_t^{GRU}. This is then passed through a second Layer Norm unit to ensure a normalised output.

- Finally, the h_t^{GCN} and h_t^{GRU} representations are concatenated together, before being passed through a linear layer and a leaky ReLU activation function to create the final representation for the vertex h_t^{TNA}. Inspired by residual connections often used in computer vision networks [32], this enables the model to learn the optimum mix of topological and temporal information.

The layer-wise propagation rule of the TNA block at depth l can thus be summarised as follows for the entire graph $G_t \in G'$ with normalised adjacency matrix $\hat{A}_t$:

$$\begin{aligned} H_t^{GCN} &= GCN(\hat{A}_t, H_t^{(l-1)}) \\ H_t^{GRU} &= GRU(H_t^{GCN}, H_{t-1}^{GRU}) \\ H_t^{TNA^{(l)}} &= \sigma_{lr}\big(W_s^{(l)}\text{Concat}(H_t^{GCN}, H_t^{GRU})\big) \\ TNA(\hat{A}_t, H_t^{(l)}) &= H_t^{(l)} = H_t^{TNA^{(l)}} \end{aligned} \tag{3}$$

where $W_s^{(l)}$ represents the weight matrix used to mix the topological and temporal representations, and σ_{lr} is the leaky ReLU activation function with a negative slope of 0.01.

2) Overall Model Architecture: As with normal GCN layers, TNA blocks can be stacked to aggregate information from greater depth within a graph, with each additional block adding one extra hop from which information can be aggregated for a certain vertex. However, as our TNA blocks are recurrent, information can also be aggregated from how connectivity within these hops has evolved over time, instead of just their present state. After extensive ablation studies (detailed in Section V-A), we use the final configuration of the model detailed in Figure 3. Our model contains two stacked TNA blocks, to learn information from two hops within the temporal neighbourhood. This is then passed to two independent GCN layers which perform a final aggregation of this temporal representation. From these two layers, the final representation matrix Z_t is sampled using techniques from variational inference, specifically the reparametrisation trick [33].

Variational Sampling - To create the final representation matrix $Z_t \in \mathbb{R}^{|V_t| \times d}$, the output from the two GCN layers GCN_μ and GCN_σ are used to parametrise a unit Gaussian distribution $\mathcal{N}$, from which Z_t is then sampled, rather than being explicitly drawn. This is the same concept used in Variational Auto-Encoders [33], and has previously been demonstrated to work well for creating more robust and meaningful vertex level representations [12], [34]. Our inference model used to create the vertex representations of graph G_t, with adjacency matrix A_t and identity matrix of A_t, X_t, can thus be described as :

$$q(Z_t|X_t, A_t) = \prod_{v=1}^{|V_t|} \mathcal{N}(z_v|GCN\mu_v,\, diag(GCN\sigma_v^2)), \tag{4}$$

where q is our approximation of the true and intractable distribution we are interested in capturing – $p(A_{t+1}|Z_t)$. Here, both GCN_μ and GCN_σ take input from two stacked TNA layers as detailed in Figure 3.

Generative Model - To decode the information contained within Z_t, a generative model is created to explicitly predict the new edges appearing in the next graph in the sequence. Here, the inner-product between the latent representation is used to directly predict A_{t+1}:

$$p(A_{t+1}|Z_t) = \prod_{i=1}^{|V|}\prod_{j=1}^{|V|} p(A_{t+1_{i,j}}|\sigma_s(z_i z_j^\mathsf{T})), \tag{5}$$

where $A_{t+1_{i,j}}$ represents elements from A_{t+1} and z refers to the rows of each vertex taken from Z_t.

This generative model is one of the key advantages of our approach, as it means that we have zero learnable parameters in the decoder portion of the model. This is in contrast to many competing approaches, which often require as many parameters as in the encoder to create a decoder with the desired functionality [19]. This results in our approach being able to scale to significantly larger graphs, with longer histories than some of the competing approaches, whilst also being less prone to over-fitting to none-changing edges.

D. Objective Function

To train the TNA model, and as is common for variational methods [12], [33], we directly optimise the lower bound $\mathcal{L}$ with regards to the model parameters:

$$\mathcal{L} = \mathbb{E}_{q(Z_t|X_t,A_t)}\Big[\log p(A_{t+1}|Z_t)\Big] - KL(q(Z_t|A_t, X_t)||p(Z_t)), \tag{6}$$

where $KL()$ is the Kullback-Leibler distance between p and q. We use a Gaussian prior as the distribution for $p(Z_t)$.

In addition, we apply L_2 regularization to our model parameters to help with over-fitting, which is defined as:

$$\mathcal{L}_{reg} = \lambda \sum_{i=1}^{|\Theta|} \Theta_i^2, \tag{7}$$

where λ is a scaling factor, set to 10^{-5}. Consequently, the final objective function for our model is:

$$\mathcal{L}_{final} = \mathcal{L} + \mathcal{L}_{reg}. \tag{8}$$

E. Model Parameters and Training Procedure

After initial grid-searches, we empirically found two layers of Temporal Neighbourhood Aggregation, followed by variational sampling, to yield the optimal performance, with the first layer comprising 32 filters, whilst the second having 16 filters. For training the model, we empirically found using full-batch gradient descent with the RMSProp algorithm, a learning rate of 0.001 and 200 epochs to give the best results. Our model has been implemented in PyTorch [7].

IV. Experimental Setup

We detail the setup of our experimental evaluation, as well as the baseline approaches and the datasets we use.

A. Evaluation Overview and Methodology

As the primary goal is to create vertex representations which are better at encoding temporal change, we will be using the task of future link prediction as our primary objective. More formally, we are trying to maximise the probability of $\mathcal{P}(G_t|G_1...G_{t-1})$. In the context of machine learning, this can be defined as training a model from a temporal G' using $G_1...G_{t-1}$ such that it can predict the new edges in G_t, $E_t \setminus E_{t-1}$. The full training and evaluation process is detailed in Algorithm 1. Many recent methods attempt to solve this problem via vertex embedding similarity – i.e. vertices with more similar embeddings, according to some metric, are more likely to be connected via an edge [5], [10], [12].

Graph edges are predicted as follows: given the learned vertex embeddings, the future adjacency matrix is reconstructed via the dot product of the embedding matrix $A'_{t+1} = \sigma(Z_t Z_t^{\mathsf{T}})$. This reconstructed adjacency matrix is compared with the true graph to assess how well the embedding is able to reconstruct the future graph.

Algorithm 1: New edge predicition procedure

Input : The temporal graph $G' = \{G_1, G_2, ..., G_T\}$
Output: Mean AUC and AP scores for predicting new edges for each graph in G'

for *all* $G_t \in G'$ *where* $t \geq 3$ **do**
  Load and pre-process the graphs $G_1, G_2, ..., G_T$
  Create new model Θ_i (as shown in Figure 3)
  Train Θ_i on sequence $G_1, G_2, ..., G_{t-1}$, where each graph is the input and used to predict the following one
  Predict new edges in G_t using $\Theta_i(G_{t-1})$: $E_t \setminus E_{t-1}$
  Store AUC and AP values
end
return *Mean AUC and AP values over* G'

B. Performance Metrics

As one can consider the task of link prediction to be a binary classification problem (an edge can only be present or not), we make use of two standard binary classification metrics:

- *Area Under the Receiver Operating Characteristic Curve (AUC)* – The ratio between the True Positive Rate (TPR) and False Positive Rate (FPR) measured at various classification thresholds.
- *Mean Average Precision (AP)* – Across the set of test edges: $AP = \frac{TP}{TP+FP}$, where TP denotes the number of true positives the model predicts, and FP denotes the number of false positives.

For both of the chosen metrics, a larger value indicates more correctly predicted edges.

C. Datasets

When performing our experimental evaluation, we employ the empirical datasets presented in Table II. The graphs represent a range of domains, sizes and temporal complexities.

Bitcoin-Alpha (Bitcoina) - Representing a trust network within a platform entitled Bitcoin Alpha, where edges are formed as users interact and rate each others reputation. The graph covers a range of edges formed between 8th October 2010 and 22nd January 2016, which we partition into 62 monthly snapshots. The task of new edge prediction is thus analogous to predicting if two users are going to interact within the next month.

Wiki-Vote (Wiki) - Representing a vote of escalating user privileges between users and administrators on the Wikipedia website. The graph covers a range of edges formed between 28th March 2004 and 6th January 2008, which we partition into 34 monthly snapshots. The task of new edge prediction within this data is analogous to predicting if two users are going to vote for each other within the next week.

UCI-Messages (UCI) - Representing private messages sent between users on the University of California Irvine social network platform. The graph covers a range of edges formed between 15th April 2004 and 25th October 2004, which we

Dataset	$\|V\|$	$\|E\|$	First Edge	Last Edge	Num Snapshots	# New Edges	Reference
Bitcoin-Alpha (Bitcoina)	3,783	24,186	08/09/2010	22/01/2016	62	227	[35]
Wiki-Vote (Wiki)	7,115	103,689	28/02/2005	06/01/2008	34	2963	[35]
UC Irvine Messages (UCI)	1,899	20,296	15/04/2004	25/08/2004	27	513	[36]

TABLE II: Empirical graph datasets, where # New Edges is the average number of new edges added between time points.

partition into 27 weekly snapshots. The task of new edge prediction would represent the likelihood that two users will exchange messages with each other over the next week.

1) Synthetic Datasets: In addition, we use two synthetic datasets: a Stochastic Block Model (SBM) graph and a randomly perturbed version of the Cora dataset (R-Cora).

SBM - A random graph of 3,000 vertices, which evolves over 30 time points using the SBM algorithm [37]. The graph contains 3 communities and at each time point, 20 vertices will evolve by switching from one community to another.

R-Cora - To create this synthetic dataset, we take the original Cora dataset representing a citation network, and perturb the graph using the random rewire method [38], [39]. The rewiring process alters a given source graph's degree distribution by randomly altering the source and target of a set number of edges. During this rewiring process, it is not guaranteed that the source or target of the edge will be altered, which indeed is not always possible due to the topology of the graph. Also, the rewiring process does not change the total number of edges or vertices within the graph. We employ *Erdős* rewiring, i.e. the resulting topology of the graph begins to resemble a Erdős-Rényi graph, where the edges are uniformly distributed between vertices.

D. Baseline Approaches

We compare our approach against a variety of state-of-the-art graph representation learning techniques, both static and dynamic. We choose the baselines which compare most directly with our proposed approach, meaning we opt for comparators which take advantage of deep neural networks to create vertex embeddings.

- *GAE* [12]: A non-probabilistic Graph Convolutional Auto-encoder (GAE), where the model is trained on G_{t-1} and then directly predicts new edges in G_t.
- *GVAE* [12]: A Graph Variational Convolutional Auto-encoder (GVAE), trained in the same manner as the GAE.
- *TO-GAE* [34]: A GAE model training procedure which enables temporal offset reconstruction, where the model is trained on G_{t-2} to predict G_{t-1}. G_{t-1} is subsequently used as input and the ability to predict G_t is measured.
- *TO-GVAE* [34]: A GVAE model trained using the temporal offset reconstruction method.
- *DynAE* [19]: A non-convolutional graph embedding model, similar to SDNE [17], extended to temporal graphs by concatenating the rows of the past graphs together before being passed into the model.
- *DynRNN* [19]: A non-convolutional graph embedding model, where stacked LSTM units are used to encode the temporal graph directly. The approach also requires a decoder model, also comprised of stacked LSTM units, to reconstruct the next graph from the embedding.
- *DynAERNN* [19][2]: A combination of the previous two models, where a dense auto-encoder is used to learn a compressed representation which is passed to stacked LSTM units for temporal learning. It requires a large decoder, with both dense and LSTM layers, to predict the next graph. The E-LSTM-D approach [41] is also extremely similar to this model.
- *D-GCN:* [20], [21]: A dynamic GCN, similar to approaches proposed in [20] and [21]. Here, three stacked GCN layers are used to capture structural information with an LSTM unit used to learn temporal information and produce the final embeddings. To directly predict the next graph, we use an inner-product decoder on the embedding matrix.

We attempted to compare with GCN-GAN [22] and GC-LSTM [23], but we were unable to get them to scale to the size of graphs we are using for our experimentation.

E. Experimental Environment

Experimentation was performed on a system with 2 * NVIDIA Titan Xp GPUs, 2.3GHz Intel Xeon E5-2650 v3, 64GB RAM, with Ubuntu Server 18.04 LTS, Python 3.7, CUDA 10.1, CuDNN v7.4 and PyTorch 1.1.

V. Results

We evaluate our TNA approach using comparisons against state-of-the-art approaches and ablation studies using well-established datasets (Section IV-C).

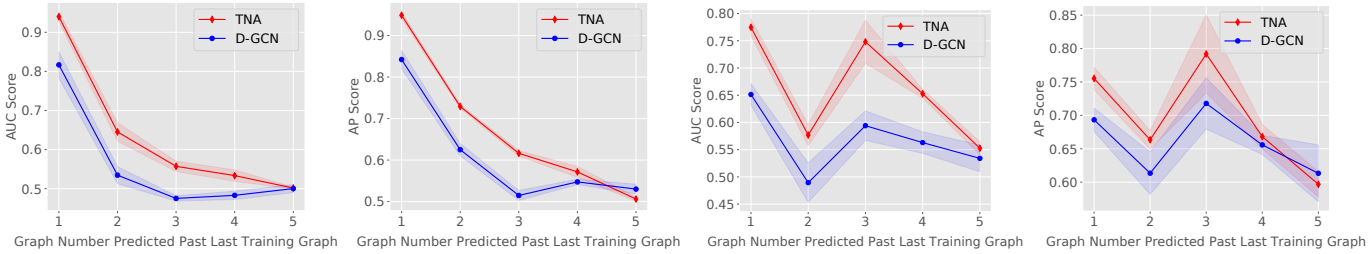
A. Ablation Study

One of the major contributions of our work is highlighting how each component of our TNA model is crucial in producing good temporal embeddings. To highlight this, Table IV shows how adding components of the model sequentially affects the performance of predicting new edges in the final graph of the Bitcoina dataset. It is important to note that adding temporal information from both the first and second hop neighbourhood (Model TTV) lifts both AUC and AP scores by approximately 10% versus just first hop temporal information (Model TGV). This supports our hypothesis that a vertex requires temporal information from more than just its first-order neighbourhood in order to predict future edges. The ablation study also demonstrates that, with a modest increase in the number of parameters, the temporal models are able to exploit the rich information available in the graph's past evolution to much more accurately predict future edges.

[2]For the Dyn* family of algorithms, we use the implementations as provided by the authors as part of their DynamicGEM package [40].

Dataset	Approach	AUC			AP			$\lvert\Theta\rvert$
		25%	50%	100%	25%	50%	100%	
Bitcoina	GAE	0.466 ± 0.025	0.497 ± 0.042	0.531 ± 0.127	0.613 ± 0.031	0.643 ± 0.042	0.681 ± 0.093	$121K$
	GVAE	0.577 ± 0.048	0.602 ± 0.046	0.620 ± 0.083	0.634 ± 0.043	0.654 ± 0.040	0.670 ± 0.068	$122K$
	TO-GAE	0.551 ± 0.053	0.566 ± 0.053	0.576 ± 0.124	0.694 ± 0.038	0.701 ± 0.038	0.715 ± 0.085	$120K$
	TO-GVAE	0.598 ± 0.048	0.620 ± 0.045	0.631 ± 0.081	0.646 ± 0.044	0.665 ± 0.040	0.631 ± 0.081	$122K$
	DynAE	0.281 ± 0.080	0.247 ± 0.065	0.209 ± 0.071	0.435 ± 0.012	0.442 ± 0.012	0.439 ± 0.023	$4.16M$
	DynRNN	0.181 ± 0.081	0.170 ± 0.059	0.155 ± 0.066	0.388 ± 0.014	0.388 ± 0.011	0.393 ± 0.022	$69.9M$
	DynAERNN	0.093 ± 0.090	0.071 ± 0.066	0.048 ± 0.054	0.326 ± 0.022	0.320 ± 0.016	0.318 ± 0.012	$6.98M$
	D-GCN	0.622 ± 0.084	0.572 ± 0.080	0.519 ± 0.144	0.697 ± 0.058	0.661 ± 0.058	0.623 ± 0.107	$125K$
	TNA	$\mathbf{0.665 \pm 0.067}$	$\mathbf{0.698 \pm 0.075}$	$\mathbf{0.775 \pm 0.110}$	$\mathbf{0.762 \pm 0.048}$	$\mathbf{0.792 \pm 0.054}$	$\mathbf{0.849 \pm 0.079}$	$133K$
UCI	GAE	0.561 ± 0.075	0.600 ± 0.075	0.606 ± 0.092	0.661 ± 0.066	0.688 ± 0.060	0.689 ± 0.079	$61K$
	GVAE	0.571 ± 0.079	0.606 ± 0.074	0.619 ± 0.065	0.585 ± 0.059	0.621 ± 0.063	0.625 ± 0.060	$62K$
	TO-GAE	0.601 ± 0.059	0.633 ± 0.061	0.625 ± 0.087	0.682 ± 0.053	0.705 ± 0.050	0.699 ± 0.076	$61K$
	TO-GVAE	0.582 ± 0.072	0.614 ± 0.069	0.624 ± 0.062	0.590 ± 0.057	0.624 ± 0.062	0.627 ± 0.060	$62K$
	DynAE	0.234 ± 0.066	0.168 ± 0.076	0.128 ± 0.067	0.436 ± 0.019	0.435 ± 0.021	0.433 ± 0.017	$2.28M$
	DynRNN	0.161 ± 0.019	0.176 ± 0.024	0.159 ± 0.048	0.365 ± 0.016	0.370 ± 0.016	0.369 ± 0.029	$21.8M$
	DynAERNN	0.033 ± 0.032	0.021 ± 0.025	0.013 ± 0.019	0.314 ± 0.005	0.312 ± 0.004	0.312 ± 0.003	$4.15M$
	D-GCN	0.508 ± 0.041	0.555 ± 0.071	0.565 ± 0.068	0.605 ± 0.045	0.653 ± 0.066	0.656 ± 0.072	$64K$
	TNA	$\mathbf{0.694 \pm 0.077}$	$\mathbf{0.749 \pm 0.073}$	$\mathbf{0.764 \pm 0.071}$	$\mathbf{0.702 \pm 0.073}$	$\mathbf{0.763 \pm 0.075}$	$\mathbf{0.783 \pm 0.067}$	$72K$
Wiki	GAE	0.491 ± 0.035	0.487 ± 0.038	0.502 ± 0.040	0.642 ± 0.029	0.621 ± 0.033	0.617 ± 0.032	$228K$
	GVAE	0.580 ± 0.024	0.573 ± 0.018	0.563 ± 0.024	0.598 ± 0.032	0.589 ± 0.025	0.572 ± 0.029	$229K$
	TO-GAE	0.537 ± 0.052	0.556 ± 0.049	0.552 ± 0.048	0.700 ± 0.032	0.697 ± 0.027	0.668 ± 0.044	$228K$
	TO-GVAE	0.599 ± 0.028	0.595 ± 0.021	0.579 ± 0.029	0.613 ± 0.036	0.604 ± 0.029	0.583 ± 0.034	$229K$
	DynAE	0.354 ± 0.034	0.325 ± 0.041	0.244 ± 0.089	0.448 ± 0.009	0.463 ± 0.016	0.467 ± 0.013	$7.5M$
	DynAERNN	0.183 ± 0.024	0.179 ± 0.026	0.127 ± 0.056	0.342 ± 0.005	0.341 ± 0.006	0.329 ± 0.012	$11.9M$
	D-GCN	0.628 ± 0.160	0.591 ± 0.115	0.563 ± 0.087	0.745 ± 0.104	0.686 ± 0.094	0.629 ± 0.089	$231K$
	TNA	$\mathbf{0.674 \pm 0.034}$	$\mathbf{0.644 \pm 0.044}$	$\mathbf{0.634 \pm 0.050}$	$\mathbf{0.759 \pm 0.025}$	$\mathbf{0.740 \pm 0.032}$	$\mathbf{0.736 \pm 0.039}$	$239K$

TABLE III: Next graph prediction results presented as mean values with standard deviation when predicting at various percentages of the length of the time-sequence. A bold value indicates the highest score for that metric. The number of parameters required by each model for the specific datasets are also included.



(a) AUC score on Wiki (b) AP score on Wiki (c) AUC score on UCI (d) AP score on UCI

Fig. 4: AUC and AP for the Wiki and UCI datasets when predicting new edges n number of time points away from the end of the training sequence. Results presented as the mean of three uniquely trained models, each with a different random seed.

Approach	AUC	AP	$\lvert\Theta\rvert$
GGG	0.574	0.747	$121K$
GGV	0.721	0.705	$122K$
TGV	0.772	0.809	$130K$
TTV	0.863	0.916	$132K$
TTV/LN	0.927	0.932	$132K$
TTV/LN/SC (TNA)	**0.977**	**0.976**	$133K$

TABLE IV: Ablation study results on the Bitcoina dataset. G is a GCN layer, V is a varitonal sampling layer, T is a GCN + GRU layer, LN is Layer Norm and SC is a skip-connection. $\lvert\Theta\rvert$ is the total number of learnable parameters in the model.

B. Next Graph Link Prediction

As the main focus of our model, we present results for predicting new edges in the next temporal graph, using the procedure detailed in Algorithm 1, in Table III[3]. The table shows that TNA significantly outperforms the baseline approaches when predicting new edges in the next graph at all points along the time series. Compared with the Dyn* family of approaches, it is striking to note the significant number of parameters required by the models (often well over an order of magnitude more) and their poor performance in predicting new edges. We believe it is highly likely that this family of models is using the extra parameters to over-fit to the edges that do not change over time, resulting in bad predictive capability for the ones that do. It is also interesting to note that, compared with the D-GCN approach, TNA is better able to capture the dependences needed for good long-term prediction. For two

[3]DynRNN is missing for the Wiki dataset as it could not fit in GPU memory.

Dataset	Approach	AUC	AP
SBM	GAE	$\mathbf{0.505 \pm 0.018}$	0.451 ± 0.009
	GVAE	0.500 ± 0.012	$\mathbf{0.503 \pm 0.011}$
	TO-GAE	0.504 ± 0.017	0.451 ± 0.008
	TO-GVAE	0.500 ± 0.012	0.503 ± 0.011
	DynAE	0.023 ± 0.003	0.431 ± 0.008
	DynRNN	0.039 ± 0.005	0.348 ± 0.009
	DynAERNN	0.008 ± 0.000	0.308 ± 0.000
	D-GCN	0.458 ± 0.017	0.458 ± 0.017
	TNA	0.502 ± 0.024	0.502 ± 0.017
R-Cora	GAE	0.501 ± 0.015	0.500 ± 0.0100
	GVAE	0.491 ± 0.011	0.494 ± 0.002
	TO-GAE	0.500 ± 0.013	$\mathbf{0.502 \pm 0.009}$
	TO-GVAE	0.490 ± 0.011	0.494 ± 0.011
	DynAE	0.356 ± 0.001	0.479 ± 0.003
	DynRNN	0.308 ± 0.011	0.381 ± 0.011
	DynAERNN	0.201 ± 0.000	0.346 ± 0.000
	D-GCN	$\mathbf{0.502 \pm 0.011}$	0.500 ± 0.008
	TNA	0.493 ± 0.012	0.493 ± 0.012

TABLE V: Next graph prediction results on sythnetic graphs presented as mean values with standard deviation when predicting at each point in the time series.

datasets our model improves the past graph evolution data it has to learn from. This is demonstrated by the increasing AUC and AP scores for the Bitcoina and UCI datasets. However, all approaches struggle on the synthetic datasets due to the inherent random nature, as seen in Table V.

C. Full Graph Reconstruction

To measure the ability of the representations learned by the TNA model to be used as general purpose embeddings, we look at the problem of future graph reconstruction. Here, the performance of the model at predicting the presence of edges in the full graph G_t (given $G_1..G_{t-1}$) is measured – highlighting how we do not sacrifice performance at predicting existing edges. This will allow us to investigate the ability of the model to predict not only new edges, but that existing edges have not been removed. As before, a new model is trained to predict the final graph in the sequence given all previous time points, with the final results presented as the mean over all graphs in the sequence. However, instead of predicting edges which have appeared since the last time point, here the results are for a balanced set of random sampled positive and negative edges in E_t which may or may not include ones formed since the previous time point.

The results for this experiment are presented in Table VI where for the sake of brevity, we compare with only the temporal baselines. It is obvious that many of the baselines, especially the Dyn* family of approaches perform much better at predicting existing edges than new ones. This further suggests that they are utilising their larger set of parameters to, in some way, over-fit to edges which have been in the graph for a longer length of time, which form the vast majority. However despite this, our TNA approach still performs well at this task, displaying comparable performance with the baseline approaches and even outperforming them on the Wiki dataset. This further strengthens the argument that having recurrence at each hop in the neighbourhood aggregation produces a better representation, whilst requiring fewer parameters.

Dataset	Approach	AUC	AP
Bitcoina	DynAE	0.830 ± 0.068	0.844 ± 0.050
	DynRNN	0.922 ± 0.059	0.937 ± 0.039
	DynAERNN	$\mathbf{0.968 \pm 0.057}$	$\mathbf{0.981 \pm 0.034}$
	D-GCN	0.919 ± 0.021	0.934 ± 0.016
	TNA	0.932 ± 0.024	0.945 ± 0.018
UCI	DynAE	0.905 ± 0.061	0.908 ± 0.055
	DynRNN	0.957 ± 0.015	0.954 ± 0.010
	DynAERNN	$\mathbf{0.988 \pm 0.014}$	$\mathbf{0.993 \pm 0.009}$
	D-GCN	0.829 ± 0.019	0.862 ± 0.014
	TNA	0.821 ± 0.015	0.847 ± 0.012
Wiki	DynAE	0.765 ± 0.088	0.795 ± 0.062
	DynAERNN	0.882 ± 0.072	0.934 ± 0.037
	D-GCN	0.905 ± 0.019	0.936 ± 0.015
	TNA	$\mathbf{0.919 \pm 0.014}$	$\mathbf{0.945 \pm 0.007}$

TABLE VI: Results for predicting both new and old edges in the final graph in the sequence, presented as a mean and standard deviation over the whole time sequence. A bold value indicates the highest score for that metric. TNA remains competitive with, and even beats many baseline approaches with a much greater number of parameters.

D. Future Graph Evolution

For our final experiment, we investigate how TNA performs when predicting new edges further into the future than the next graph. We train the models on 70% of the available temporal history, then predict new edges and compare with the remaining ground truth data. To achieve this, we feed the graph predicted by the models as the next graph in the sequence back into the model, which is subsequently used to predict the next graph. This is similar to using RNNs as generative models to produce text data [42] and can be seen as a combination of both the previous tasks. Figure 4 displays the results for this task, where we compare with the closet baseline from Section V-B. The results show how TNA is better able to predict new edges into the future, emphasising its capability to learn a good temporal representation for the vertices.

VI. Conclusion

Many real-world graph datasets have rich and complex temporal information available which is disregard by the majority of the current approaches for creating vertex representations. In this paper, we have introduced the Temporal Neighbourhood Aggregation model for representation learning on large, complex temporal graphs. Our approach demonstrates excellent performance through extensive experimental evaluation, beating several competing temporal and static models, when predicting future edges not seen in the training data. The TNA model can learn complex temporal patterns present at multiple depths within a vertices neighbourhood, creating the final vertex representation via the use of variational sampling.

For future work, we will investigate replacing the GCN in our model with an approach designed for inductive learning

[30] to allow for training on even larger graph datasets, as well as enabling vertex arrival to be modelled. We also plan to experiment using the learned representations for additional tasks, such as temporal classification.

ACKNOWLEDGEMENT

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GPU used for this research. Additionally we thank the Engineering and Physical Sciences Research Council UK (EPSRC) for funding.

REFERENCES

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[2] R. Yang, Y. Bai, Z. Qin, and T. Yu, "Egonet: identification of human disease ego-network modules," *BMC genomics*, vol. 15, no. 1, p. 314, 2014.

[3] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chemical science*, vol. 9, no. 2, pp. 513–530, 2018.

[4] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende *et al.*, "Interaction networks for learning about objects, relations and physics," in *Advances in Neural Information Processing Systems*, 2016, pp. 4502–4510.

[5] A. Grover and J. Leskovec, "node2vec : scalable feature learning for networks," *International Conference on Knowledge Discovery and Data Mining*, 2016.

[6] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: a survey," *arXiv preprint arXiv:1705.02801*, 2017.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[8] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," *Advances in neural information processing systems*, pp. 585–591, 2002.

[9] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," *International conference on World Wide Web*, pp. 37–48, 2013.

[10] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: online learning of social representations," *International Conference on Knowledge Discovery and Data Mining*, 2014.

[11] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on Unsupervised and Transfer Learning*, 2012, pp. 37–49.

[12] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[13] S. Pandhre, H. Mittal, M. Gupta, and V. N. Balasubramanian, "Stwalk: learning trajectory representations in temporal graphs," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*. ACM, 2018, pp. 210–219.

[14] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *Iternational Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2672–2681.

[15] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *3rd International Workshop on Learning Representations for Big Networks (WWW BigNet)*, 2018.

[16] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," *arXiv preprint arXiv:1805.11273*, 2018.

[17] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1225–1234.

[18] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," *arXiv preprint arXiv:1511.05641*, 2015.

[19] P. Goyal, S. R. Chhetri, and A. Canedo, "dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowledge-Based Systems*, 2019.

[20] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *Pattern Recognition*, p. 107000, 2019.

[21] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *International Conference on Neural Information Processing*. Springer, 2018, pp. 362–373.

[22] K. Lei, M. Qin, B. Bai, G. Zhang, and M. Yang, "Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 388–396.

[23] J. Chen, X. Xu, Y. Wu, and H. Zheng, "Gc-lstm: Graph convolution embedded lstm for dynamic link prediction," *arXiv preprint arXiv:1812.04206*, 2018.

[24] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, and C. E. Leisersen, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," *arXiv preprint arXiv:1902.10191*, 2019.

[25] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. Li, "Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction," in *AAAI Conference on Artificial Intelligence*, 2019.

[26] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[29] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

[30] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[31] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[33] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[34] S. Bonner, J. Brennan, I. Kureshi, G. Theodoropoulos, A. S. McGough, and B. Obara, "Temporal graph offset reconstruction: Towards temporally robust graph representation learning," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 3737–3746.

[35] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[36] J. Kunegis, "Konect: the koblenz network collection," in *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 2013, pp. 1343–1350.

[37] B. Karrer and M. E. Newman, "Stochastic blockmodels and community structure in networks," *Physical review E*, vol. 83, no. 1, p. 016107, 2011.

[38] S. Bonner, J. Brennan, G. Theodoropoulos, I. Kureshi, and A. S. McGough, "Deep topology classification: A new approach for massive graph classification," in *International Conference on Big Data*. IEEE, 2016, pp. 3290–3297.

[39] S. Bonner, J. Brennan, I. Kureshi, M. Stephen, and G. Theodoropoulos, "Efficient comparison of massive graphs through the use of 'graph fingerprints'," in *KDD Workshop on Mining and Learning with Graphs (MLG)*, 2016.

[40] P. Goyal, S. R. Chhetri, N. Mehrabi, E. Ferrara, and A. Canedo, "Dynamicgem: A library for dynamic graph embedding methods," *arXiv preprint arXiv:1811.10734*, 2018.

[41] J. Chen, J. Zhang, X. Xu, C. Fu, D. Zhang, Q. Zhang, and Q. Xuan, "E-lstm-d: A deep learning framework for dynamic network link prediction," *arXiv preprint arXiv:1902.08329*, 2019.

[42] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.