Neural Latent Space Model for Dynamic Networks and Temporal Knowledge Graphs

Tony Gracious,[1*] Shubham Gupta,[1*] Arun Kanthali,[1] Rui M. Castro,[2] Ambedkar Dukkipati[1]
[1] Indian Institute of Science, Bangalore - 560012, INDIA.
[2] Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands.
[tonygracious, shubhamg, ambedkar]@iisc.ac.in, rmcastro@tue.nl

Abstract

Although static networks have been extensively studied in machine learning, data mining, and AI communities for many decades, the study of dynamic networks has recently taken center stage due to the prominence of social media and its effects on the dynamics of social networks. In this paper, we propose a statistical model for dynamically evolving networks, together with a variational inference approach. Our model, Neural Latent Space Model with Variational Inference, encodes edge dependencies across different time snapshots. It represents nodes via latent vectors and uses interaction matrices to model the presence of edges. These matrices can be used to incorporate multiple relations in heterogeneous networks by having a separate matrix for each of the relations. To capture the temporal dynamics, both node vectors and interaction matrices are allowed to evolve with time. Existing network analysis methods use representation learning techniques for modelling networks. These techniques are different for homogeneous and heterogeneous networks because heterogeneous networks can have multiple types of edges and nodes as opposed to a homogeneous network. Unlike these, we propose a unified model for homogeneous and heterogeneous networks in a variational inference framework. Moreover, the learned node latent vectors and interaction matrices may be interpretable and therefore provide insights on the mechanisms behind network evolution. We experimented with a single step and multi-step link forecasting on real-world networks of homogeneous, bipartite, and heterogeneous nature, and demonstrated that our model significantly outperforms existing models.

1 Introduction

Network analysis is by no means a new field and consequently, a towering wealth of literature that explores various aspects of network analysis is available (Goldenberg et al. 2010). With the advent of deep learning, network analysis methods more recently started focusing on representation learning techniques. These methods learn finite vector representations or embeddings for nodes and edges that can be used for downstream tasks like link prediction (Grover and Leskovec 2016), node classification (Sen et al. 2008), community detection (Fortunato 2010), and so on. These techniques are of two kinds. The first kind is for homogeneous networks with only a single type of nodes and edges. The second kind is for heterogeneous networks with multiple types of nodes and edges. These networks can also be viewed as Knowledge Graphs (KG) with different types of nodes as entities and different types of edges as relations between entities.

Early works on homogeneous network representation learning use random walk models to capture the neighborhood context (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016). The embeddings are learned so that nodes closer to each other have similar embeddings. In a KG, real-world facts are stored using edges which are represented as a triplet of the form `(Subject Entity, Relation, Object Entity)`. Here, `Subject Entity` and `Object Entity` form the nodes, and `Relation` is the type of edge. The modelling techniques developed for homogeneous networks are not applicable for KGs as the entities connected via a relation may not be similar. For example, consider the edge `(Barack Obama, Born in , USA)` which represents a relation `Born in` between the entities `Barack Obama` and `USA`. Here, entities are not similar because `Barack Obama` is a person and `USA` is a country. For representation learning in such heterogeneous networks, entities and relations are given a finite representation which is then used as input to a scoring function as in (Bordes et al. 2013; Dettmers et al. 2018; Sun et al. 2019). The embeddings are learned so that the observed edges in the KG get a higher score as compared to the unobserved edges.

One aspect that is often ignored by the existing methods is that most real-world networks evolve with time. For example, in social networks, new friendship links are formed or broken with time. Similar observations can be made in KGs as there are relations with temporal properties. For example, `(Barack Obama, President of, USA)` is valid only between 2009 and 2017. KGs which encode such facts are called temporal KGs. Here, the links change with time. Owing to many practical applications, it is important to integrate both dynamic and heterogeneous information while modelling networks.

Existing methods for dynamic homogeneous networks have used techniques such as temporal regularization loss (Zhou et al. 2018; Goyal et al. 2018) and RNN based se-

[*]Equal contribution.

quential architectures (Goyal, Chhetri, and Canedo 2020) for modelling the graph evolution. For modelling temporal KGs, previous methods have used temporal point process models (Trivedi et al. 2017, 2019), or a RNN based interaction history encoding model (Jin et al. 2019) to predict the future evolution of the networks. To the best of our knowledge, ours is the first method that considers homogeneous and heterogeneous networks in an unified manner.

We propose a statistical model, called Neural Latent Space Model (NLSM), for dynamic networks to address the above-mentioned challenges. If a network has R types of relations, our model uses R sets of interaction matrices, one for each type of relation. For homogeneous networks $R = 1$ and for heterogeneous networks $R > 1$. Our approach uses an unified probabilistic framework which can scale up to the complexities in the network structure. In theory, the parameters of this proposed model could be estimated using training data via Bayesian inference. However, the likelihood structure of the model is complex and non-convex, making such methods computationally infeasible. This motivates a neural network-based variational inference procedure yielding an end-to-end trainable architecture that can be used for efficient and scalable inference.

Our main contributions are as follows. **(i)** We have proposed a new statistical model for dynamic networks that encodes temporal edge dependencies and can model both homogeneous and heterogeneous networks. **(ii)** We have provided ample empirical evidence to demonstrate that our model is suitable for link forecasting and it may simultaneously provide important insights into the network evolution mechanics via interpretable embeddings. **(iii)** In dynamic homogeneous networks, we observed an average performance improvement (over existing state-of-the-art) of 4% in Micro-AUC metric for single-step link forecasting. Similarly, in dynamic bipartite networks, an average performance improvement of 8.7% in Micro-AUC metric for single-step link forecasting was observed. In dynamic heterogeneous networks, the average improvement is 7.9% in the mean reciprocal rank metric for multi-step link forecasting task.

2 Neural Latent Space Model

2.1 Modeling Individual Snapshots

In our model, time $t \in \{1, 2, \ldots, T\}$ is discrete. The network evolution is therefore described by the corresponding network snapshots at each time-step, specified by binary adjacency matrices $\mathbf{A}_r^{(t)} \in \{0, 1\}^{N \times N}$, where N is the number of nodes in the network, $r \in \{1, 2, \ldots, R\}$ is the relation between the nodes, and t denotes the time. We begin by discussing the case of homogeneous networks ($R = 1$). The extension to heterogeneous networks ($R \geq 2$) is then straightforward and we present it in Section 2.3. To avoid cluttering the notation, we drop the subscript in $\mathbf{A}_r^{(t)}$ when $R = 1$. We further assume that there are no self-loops. Each node is modeled by K attributes whose values lie in the interval $[0, 1]$. These attributes can change over time. The latent vector $\mathbf{z}_n^{(t)} \in [0, 1]^K$ is used to denote the attributes for node n at time t.

The interaction between latent vectors of each pair of nodes directly dictates the probability of observing an edge between them. For simplicity, our interaction model encodes only interactions between attributes of the same type, described by *interaction matrices*. For homogeneous networks let $\mathbf{\Theta}_k^{(t)} \in \mathbb{R}^{2\times 2}$ be a matrix that encodes the affinity between nodes with respect to attribute k at time t. At the time t, the node latent vector and interaction matrices fully determine the probability of edges being present. Formally, given $\mathbf{\Theta}_k^{(t)}$, $k = 1, \ldots, K$ and the latent vectors for all nodes $\mathbf{z}_n^{(t)}$, $n = 1, \ldots, N$, edges occur independently and the probability of an edge from node i to node j is modeled as:

$$P\left(a_{ij}^{(t)} = 1 | \mathbf{z}_i^{(t)}, \mathbf{z}_j^{(t)}, \{\mathbf{\Theta}_k^{(t)}\}_{k=1}^K\right) = \sigma\left(\sum_{k=1}^{K} \tilde{\theta}_k^{(t)}(i, j)\right), \tag{1}$$

where, $\tilde{\theta}_k^{(t)}(i, j)$ is defined as:

$$\tilde{\theta}_k^{(t)}(i, j) = \mathsf{E}_{x\sim B(z_{ik}^{(t)}), y\sim B(z_{jk}^{(t)})}\left[\Theta_k^{(t)}(x, y)\right]. \tag{2}$$

Here $\sigma(.)$ is the *sigmoid* function, $B(\alpha)$ refers to a Bernoulli distribution with parameter α and $\Theta_k^{(t)}(x, y)$ is the $(x, y)^{th}$ entry of $\Theta_k^{(t)}$ matrix. Note that x and y are independent in (2). This formulation allows representation of both homophilic and heterophilic interactions among nodes depending on the structure of the matrices $\Theta_k^{(t)}$. For the case of undirected graphs, the matrices $\Theta_k^{(t)}$ are symmetric.

The interaction model we consider is in the same spirit as the Multiplicative Attribute Graph (MAG) model (Kim and Leskovec 2012). Some other dynamic network models (Kim and Leskovec 2013) use the MAG model directly to represent each static network snapshot, however, in our case, we have a few differences: our node attributes are not restricted to being binary and we have a differentiable expectation operation as given in (2) instead of the non-differentiable "selection" operation given in (Kim and Leskovec 2012). These differences are crucial for one to use a neural network-based variational inference procedure.

2.2 Modeling Network Dynamics

Having described how each network snapshot is generated, it remains to describe how attributes and their interactions evolve over time. To make an analogy with genetics, each attribute type might be seen as a *gene*, and the latent vector corresponds to the *gene expression profile* of a given node. The level of expression of each attribute might change over time - nodes may start exhibiting new attributes and stop exhibiting old ones thereby leading to a change in $\mathbf{z}_n^{(t)}$. At the same time, the role of each attribute in regulating the presence of edges in the network may also change over time leading to a change in $\mathbf{\Theta}_k^{(t)}$ matrices.

One approach to model the dynamics of a network is to use domain expertise to impose a specific set of assumptions on the process governing the dynamics. However, this limits the class of networks that can be faithfully modeled. Instead, we adopt the strategy of imposing a minimal set of

assumptions on the dynamics. This is in the same spirit as in the models used in tracking using stochastic filtering (e.g., Kalman filters and variants) (Yilmaz, Javed, and Shah 2006), where dynamics are rather simple and primarily capture the insight that the state of the system cannot change too dramatically over time. The use of simple dynamics together with a powerful function approximator (a neural network) during the inference ensures that a simple yet powerful model can be learned from observed network data.

Let $\bar{\boldsymbol{\theta}}_k^{(t)}$ be a vector consisting of the entries of the $\boldsymbol{\Theta}_k^{(t)}$ matrix[1]. We model the evolution of interaction matrices as:

$$\bar{\boldsymbol{\theta}}_k^{(t)} \sim \mathcal{N}(\bar{\boldsymbol{\theta}}_k^{(t-1)}, s_\theta^2 \mathbf{I}),\ k = 1, ..., K,\ t = 2, ..., T, \quad (3)$$

independently over time, where $s_\theta^2 \in \mathbb{R}^+$ is a model hyperparameter and $\mathbf{I}$ denotes the identity matrix. This model captures the intuition that the interaction matrices will likely not change dramatically over time.

Since the entries of the latent vector $\mathbf{z}_n^{(t)}$ are restricted to lie in $[0, 1]$, a similar dynamics model as above is not possible. A simple workaround is to re-parameterize the problem by introducing the vectors $\boldsymbol{\psi}_n^{(t)} \in \mathbb{R}^K$ such that

$$z_{nk}^{(t)} = \sigma(\psi_{nk}^{(t)}). \quad (4)$$

As before, $\sigma(.)$ is the sigmoid function. Now we can have an evolution model similar to (3) on vectors $\boldsymbol{\psi}_n^{(t)}$. That is,

$$\boldsymbol{\psi}_n^{(t)} \sim \mathcal{N}(\boldsymbol{\psi}_n^{(t-1)}, s_\psi^2 \mathbf{I})\ n = 1, ..., N,\ t = 2, ..., T, \quad (5)$$

where s_ψ^2 is a model hyperparameter. This in turn models the evolution of vectors $\mathbf{z}_n^{(t)}$.

Note that (3) and (5) only imply that the values of variables are unlikely to change very quickly. Other than that, they do not place any strong or network-specific restriction on the dynamics. The hyperparameters s_θ^2 and s_ψ^2 control the radius around the current value of the variable within which it is likely to stay in the next timestep.

This approach for modeling dynamics has advantages and disadvantages. The major advantage is flexibility since during inference time, a powerful enough function approximator can learn appropriate network dynamics from the observed data. However, if we regard this proposal as a generative model, then it will lead to unrealistic global behavior. Nonetheless, locally (in time) it will capture the type of dynamics one sees in many networks, and this is enough to ensure good tracking performance. In many real-world cases, a suitable amount of observed data is available but clues about the network dynamics are unavailable. Since the task is to gain meaningful insights from the data, we believe the advantages of this approach outweigh the disadvantages.

Note that (3) and (5) are applicable from timestep 2 onward. We need a way to obtain the initial values for $\boldsymbol{\psi}_n^{(1)}$ and $\bar{\boldsymbol{\theta}}_k^{(1)}$ for $n = 1, 2, ..., N$ and $k = 1, 2, ..., K$. The initial vectors $\boldsymbol{\psi}_n^{(1)}$ and $\bar{\boldsymbol{\theta}}_k^{(1)}$ are sampled from a prior distribution. We use the following prior distributions:

$$\bar{\boldsymbol{\theta}}_k^{(1)} \sim \mathcal{N}(\mathbf{0}, \sigma_\theta^2 \mathbf{I}), \quad (6)$$

$$\boldsymbol{\psi}_n^{(1)} \sim \mathcal{N}(\mathbf{0}, \sigma_\psi^2 \mathbf{I}). \quad (7)$$

Here, σ_θ and σ_ψ are hyperparameters. In our experiments, we set these hyperparameters to a high value ($\sigma_\theta = \sigma_\psi = 10$). This allows the initial embeddings to become flexible enough to represent the first snapshot faithfully. After that, the assumption that the network changes slowly ((3) and (5)) is used to sample the value of random variables $\boldsymbol{\psi}_n^{(t)}$ and $\bar{\boldsymbol{\theta}}_k^{(t)}$ for $t = 2, 3, ..., T$.

We make the following independence assumptions: given $\boldsymbol{\psi}_n^{(t-1)}$ the vectors $\boldsymbol{\psi}_n^{(t)}$ are independent of any quantity indexed by time $t' \leq t - 1$. An analogous statement applies to the interaction matrices $\bar{\boldsymbol{\theta}}_k^{(t)}$. Finally, given $\boldsymbol{\psi}_i^{(t)}$, $\boldsymbol{\psi}_j^{(t)}$ and $\bar{\boldsymbol{\Theta}}^{(t)} = \{\bar{\boldsymbol{\theta}}_k^{(t)}\}_{k=1}^K$, the entries $a_{ij}^{(t)}$ are independent of everything else. The graphical model for NLSM is given in Appendix A in the supplementary material (Gracious et al. 2021). Algorithm 1 outlines the generative process for NLSM.

2.3 Modeling Heterogeneous Networks

In a heterogeneous network, the nodes may have different *types* of relationships between them. A classic example is a knowledge graph where, for instance, `is president of` and `lives in` relations have different semantics. The set of nodes, and hence the latent vectors $\mathbf{z}_n^{(t)}$, remains unchanged. To model different types of relations, we use relation specific interaction matrices. Hence, the interaction matrix for k^{th} attribute now depends on the relation r, and we denote this by $\boldsymbol{\Theta}_{k,r}^{(t)}$, for $k = 1, 2, \ldots, K$, and $r = 1, 2, \ldots, R$. To compute the edge probabilities under a specified relation r, (1) only uses interaction matrices of type r. Similarly, (2), (3), and (6) individually apply to each r dependent interaction matrix. Using common latent attributes for nodes captures the required relationships among relations, hence the interaction matrices can evolve independently of each other. In a graph with several types of nodes, because all nodes share the same set of attributes, K must be large enough to accommodate the diverse properties that must be encoded. Naturally, not all elements of the node vectors will be relevant to all node types.

3 Inference in NLSM

As before, to maintain readability, we describe the inference procedure for the case of $R = 1$. The general case of $R \geq 1$ trivially follows along the same lines. In practice, an observed sequence of network snapshots $\mathbf{A} = [\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(T)}]$ is available, and the main inference task is to estimate the values of the underlying latent random variables. Performing exact inference in NLSM is intractable because the computation of marginalized log probability of observed data results in integrals that are hard to evaluate. Thus, we adopt approximate inference techniques.

[1]For directed graphs the matrix $\boldsymbol{\Theta}_k^{(t)}$ can be arbitrary, therefore $\bar{\boldsymbol{\theta}}_k^{(t)}$ will have four entries. In the case of undirected graphs the matrices are symmetric, and therefore three entries suffice.

Algorithm 1 Generative process for NLSM

Input: N: Number of nodes,
K: Latent vector dimension,
T: Number of timesteps,
s_θ^2: Hyperparameter used in (3),
s_ψ^2: Hyperparameter used in (5),
σ_θ^2: Hyperparameter used in (6), and
σ_ψ^2: Hyperparameter used in (7)

Sample $\boldsymbol{\psi}_n^{(1)}$ using (7) for $n = 1, 2, ..., N$
Sample $\bar{\boldsymbol{\theta}}_k^{(1)}$ using (6) for $k = 1, 2, ..., K$
for $t = 1$ **to** $T - 1$ **do**
  Compute $\mathbf{z}_n^{(t)}$ by using $\boldsymbol{\psi}_n^{(t)}$ in (4) for $n = 1, 2, ..., N$
  Sample $a_{ij}^{(t)}$ using (1) for $i, j = 1, 2, ..., N, i \neq j$
  Sample $\boldsymbol{\psi}_n^{(t+1)}$ using (5) for $n = 1, 2, ..., N$
  Sample $\bar{\boldsymbol{\theta}}_k^{(t+1)}$ using (3) for $k = 1, 2, ..., K$
end for
Compute $\mathbf{z}_n^{(T)}$ by using $\boldsymbol{\psi}_n^{(T)}$ in (4) for $n = 1, 2, ..., N$
Sample $a_{ij}^{(T)}$ using (1) for $i, j = 1, 2, ..., N, i \neq j$
Return: $[\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(T)}]$

Our goal is to compute an approximation to the true posterior distribution $P(\{\boldsymbol{\Psi}^{(t)}, \boldsymbol{\Theta}^{(t)}\}_{t=1}^T | \{\mathbf{A}^{(t)}\}_{t=1}^T)$. Note that in our approach K, s_θ, s_ψ, σ_θ and σ_ψ are hyperparameters that are simply set by the user. We pose the inference problem as an optimization problem by using Variational Inference (Blei, Kucukelbir, and McAuliffe 2017) and parameterize the approximating distribution by a neural network. There are several benefits like efficiency and scalability (Blei, Kucukelbir, and McAuliffe 2017) associated with the use of variational inference. Also, coupled with powerful neural networks, variational inference can model complicated distributions (Kingma and Welling 2013).

The main idea behind variational inference is to approximate the posterior distribution by a suitable surrogate. Consider a general latent variable model with the set of all observed random variables $\mathbf{X}$ and the set of all latent random variables $\mathbf{H}$. The (intractable) posterior distribution $P(\mathbf{H}|\mathbf{X})$ is approximated by using a parameterized distribution $Q_{\boldsymbol{\Phi}}(\mathbf{H})$ where $\boldsymbol{\Phi}$ is the set of all the parameters of Q. One would like the distribution Q to be as *close* to the distribution $P(\mathbf{H}|\mathbf{X})$ as possible. In general, Kullback-Leibler (KL) divergence is used as a measure of similarity between the two distributions. The goal of variational inference is to find the parameters $\boldsymbol{\Phi}$ for which $\mathrm{KL}(Q_{\boldsymbol{\Phi}}(\mathbf{H})||P(\mathbf{H}|\mathbf{X}))$ is minimized. However, this optimization objective is intractable since one cannot efficiently compute $P(\mathbf{H}|\mathbf{X})$. Nevertheless one can show that maximizing the *Evidence Lower Bound Objective (ELBO)* given by

$$\mathrm{ELBO}(\boldsymbol{\Phi}) = \mathsf{E}_Q[\log P(\mathbf{X}, \mathbf{H}) - \log Q_{\boldsymbol{\Phi}}(\mathbf{H})], \quad (8)$$

is equivalent to minimizing the KL criterion (Blei, Kucukelbir, and McAuliffe 2017) (see Appendix B in the supplementary material (Gracious et al. 2021) for proof). For most models, the ELBO can be efficiently computed or approximated by imposing a suitable set of assumptions on Q as described later. We parameterize the distribution Q by a neural network and hence $\boldsymbol{\Phi}$ represents the set of parameters of that neural network in our setting.

3.1 Approximating ELBO

The latent variables in our model correspond to the elements of $\boldsymbol{\Theta}^{(t)}$ and $\boldsymbol{\Psi}^{(t)}$ for $t = 1, 2, ..., T$. The observed variables are $\mathbf{A}^{(1)}, ..., \mathbf{A}^{(T)}$. The parameter vector $\boldsymbol{\Phi}$ consists of the weights of the neural network. Following (8), we get:

$$\mathrm{ELBO}(\boldsymbol{\Phi}) = \mathsf{E}_Q\big[\log P\big(\{\mathbf{A}^{(t)}\}_{t=1}^T, \{\boldsymbol{\Psi}^{(t)}, \boldsymbol{\Theta}^{(t)}\}_{t=1}^T\big) - \log Q_{\boldsymbol{\Phi}}\big(\{\boldsymbol{\Psi}^{(t)}, \boldsymbol{\Theta}^{(t)}\}_{t=1}^T\big)\big]. \quad (9)$$

Using the independence assumptions stated in Section 2, one can write:

$$\begin{aligned}
&\log P\big(\{\mathbf{A}^{(t)}\}_{t=1}^T, \{\boldsymbol{\Psi}^{(t)}, \boldsymbol{\Theta}^{(t)}\}_{t=1}^T\big) = \\
&\sum_{n=1}^N \log P(\boldsymbol{\psi}_n^{(1)}) + \sum_{k=1}^K \log P(\bar{\boldsymbol{\theta}}_k^{(1)}) + \\
&\sum_{t=2}^T \Big(\sum_{n=1}^N \log P(\boldsymbol{\psi}_n^{(t)} | \boldsymbol{\psi}_n^{(t-1)}) + \sum_{k=1}^K \log P(\bar{\boldsymbol{\theta}}_k^{(t)} | \bar{\boldsymbol{\theta}}_k^{(t-1)}) \Big) \\
&+ \sum_{t=1}^T \sum_{i \neq j} \log P(a_{ij}^{(t)} | \boldsymbol{\psi}_i^{(t)}, \boldsymbol{\psi}_j^{(t)}, \boldsymbol{\Theta}^{(t)}). \quad (10)
\end{aligned}$$

The right hand side of (10) can be computed using (1), (3), (4), (5), (6) and (7). Following the standard practice (Blei, Kucukelbir, and McAuliffe 2017), we also assume that $Q_{\boldsymbol{\Phi}}(.)$ belongs to a mean field family of distributions, i.e. all the variables are independent under Q:

$$\begin{aligned}
Q_{\boldsymbol{\phi}}\big(\{\boldsymbol{\Psi}^{(t)}, \boldsymbol{\Theta}^{(t)}\}_{t=1}^T\big) = &\Big(\prod_{t=1}^T \prod_{n=1}^N q_{\boldsymbol{\psi}_n}^{(t)}(\boldsymbol{\psi}_n^{(t)}) \Big) \\
&\Big(\prod_{t=1}^T \prod_{k=1}^K q_{\bar{\boldsymbol{\theta}}_k}^{(t)}(\bar{\boldsymbol{\theta}}_k^{(t)}) \Big). \quad (11)
\end{aligned}$$

We model the distributions $q_{\boldsymbol{\psi}_n}^{(t)}$ and $q_{\bar{\boldsymbol{\theta}}_k}^{(t)}$ using a Gaussian distribution as given in (12) and (13) (with some abuse of notation, $\mathcal{N}$ denotes the density of a normal distribution[2]).

$$q_{\boldsymbol{\psi}_n}^{(t)}(\boldsymbol{\psi}_n^{(t)}) = \mathcal{N}(\boldsymbol{\psi}_n^{(t)} | \mathbf{m}_{\boldsymbol{\psi}_n}^{(t)}, (\boldsymbol{\sigma}_{\boldsymbol{\psi}_n}^{(t)})^2 \mathbf{I}), \text{and} \quad (12)$$

$$q_{\bar{\boldsymbol{\theta}}_k}^{(t)}(\bar{\boldsymbol{\theta}}_k^{(t)}) = \mathcal{N}(\bar{\boldsymbol{\theta}}_k^{(t)} | \mathbf{m}_{\bar{\boldsymbol{\theta}}_k}^{(t)}, (\boldsymbol{\sigma}_{\bar{\boldsymbol{\theta}}_k}^{(t)})^2 \mathbf{I}). \quad (13)$$

Here $(\boldsymbol{\sigma}_x^{(t)})^2 \mathbf{I} = \mathrm{diag}\big((\sigma_x^{(t)})_1^2, ..., (\sigma_x^{(t)})_{|x|}^2\big)$. We wish to learn the mean and covariance parameters of Gaussian distributions in (12) and (13) (these are called *variational parameters*). There are two possible approaches for doing this: **(i)** $\mathrm{ELBO}(\boldsymbol{\Phi})$ can be directly optimized as a function of variational parameters or **(ii)** One can model the variational parameters as outputs of some other parametric function (like a neural network) and then optimize the parameters of that

[2]Define $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi|\boldsymbol{\Sigma}|)^{|\boldsymbol{\mu}|/2}} \exp\left(-\frac{1}{2} d(\mathbf{x}, \boldsymbol{\mu})\right)$, where $d(\mathbf{x}, \boldsymbol{\mu}) = (\mathbf{x} - \boldsymbol{\mu})^\intercal \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})$

parametric function. The second approach can be viewed as a form of regularization where the space in which variational parameters can lie is constrained to the range of the parametric function in use. We adopt the latter approach, and obtain the variational parameters as outputs of neural networks. We use $\mathbf{\Phi}$ to denote the set of neural network parameters. Thus $q^{(t)}_{\boldsymbol{\psi}_n}(\boldsymbol{\psi}^{(t)}_n) \equiv q^{(t)}_{\boldsymbol{\psi}_n}(\boldsymbol{\psi}^{(t)}_n; \mathbf{\Phi})$, but we do not explicitly mention the dependence on $\mathbf{\Phi}$ in general to avoid further notational clutter. The ELBO($\mathbf{\Phi}$) can now be computed by using (10) and (11) in (9). Integration of the term involving $\log P(a^{(t)}_{ij} | \boldsymbol{\psi}^{(t)}_i, \boldsymbol{\psi}^{(t)}_j, \mathbf{\Theta}^{(t)})$ is hard, so for this term we use Monte-Carlo estimation. In all our experiments we use only one sample to get an approximation to (9) as also proposed in (Kingma and Welling 2013). Additionally, we observed in our experiments that for $t = 1$, using $\mathbf{m}^{(1)}_{\boldsymbol{\psi}_n}$ and $\bar{\boldsymbol{\theta}}^{(1)}_k$ directly as a sample for Monte-Carlo estimation improves the performance for link forecasting and hence we do this in all our experiments.

3.2 Network Architecture

We use a neural network to parameterize the distributions in (12) and (13). Our network consists of four GRUs (Cho et al. 2014), one each for the mean and covariance parameters ($\mathbf{m}_{\boldsymbol{\psi}}$, $\boldsymbol{\sigma}_{\boldsymbol{\psi}}$, $\mathbf{m}_{\bar{\boldsymbol{\theta}}}$ and $\boldsymbol{\sigma}_{\bar{\boldsymbol{\theta}}}$). We refer to these GRUs as $\mathcal{G}^{\boldsymbol{\psi}}_{\mathbf{m}}$, $\mathcal{G}^{\boldsymbol{\psi}}_{\boldsymbol{\sigma}}$, $\mathcal{G}^{\bar{\boldsymbol{\theta}}}_{\mathbf{m}}$ and $\mathcal{G}^{\bar{\boldsymbol{\theta}}}_{\boldsymbol{\sigma}}$ respectively. These GRUs interact with each other only during the computation of ELBO($\mathbf{\Phi}$) since their outputs are used to compute (9). See Appendix A in the supplementary material for a visual description.

For brevity of exposition, we will only describe the inputs and outputs for $\mathcal{G}^{\boldsymbol{\psi}}_{\mathbf{m}}$. Similar ideas have been employed for other GRUs. For $t = 1, 2, ..., T-1$, $\mathcal{G}^{\boldsymbol{\psi}}_{\mathbf{m}}$ generates $\mathbf{m}^{(t+1)}_{\boldsymbol{\psi}_n}$ at timestep t for all nodes in the current batch as output. In GRUs, the output of current timestep is used as the input hidden state for the next timestep, thus the input hidden state at timestep t corresponds to $\mathbf{m}^{(t)}_{\boldsymbol{\psi}_n}$. To be consistent with this, the initial hidden state of $\mathcal{G}^{\boldsymbol{\psi}}_{\mathbf{m}}$ is set to $\mathbf{m}^{(1)}_{\boldsymbol{\psi}_n}$. This means that the initial hidden state for $\mathcal{G}^{\boldsymbol{\psi}}_{\mathbf{m}}$ is a learnable vector.

In all our experiments, we use an all 0's input vector for $\mathcal{G}^{\boldsymbol{\psi}}_{\mathbf{m}}$ at each timestep. If observable features of nodes (that may be dynamic themselves) are available, one can instead use these features as input. For $\mathcal{G}^{\boldsymbol{\psi}}_{\boldsymbol{\sigma}}$ and $\mathcal{G}^{\bar{\boldsymbol{\theta}}}_{\boldsymbol{\sigma}}$, instead of computing the variance terms, which are constrained to be positive, we compute log of variance (this is again standard practice (Kingma and Welling 2013)).

Once the mean and covariance parameters are available, we use the reparameterisation trick (Kingma and Welling 2013) to sample $\boldsymbol{\psi}^{(t)}_n$ and $\bar{\boldsymbol{\theta}}^{(t)}_k$ using (12) and (13) which are then used to approximate ELBO($\mathbf{\Phi}$) using (9) as described in Section 3.1. The training objective is to maximize ELBO($\mathbf{\Phi}$). The beauty of our model is that ELBO($\mathbf{\Phi}$) is differentiable with respect to $\mathbf{\Phi}$ and gradients can be easily computed by back-propagation. This means that one can optimize this function using a gradient-based method and therefore capitalize on the powerful optimization methods used for training neural networks. Furthermore, since ELBO uses only pairwise interactions among nodes, we can operate in a batch setting where only a subset of all nodes and the interactions within this subset are considered. This allows us to scale up to rather large networks by training our model on random batches of nodes and their sub-graphs.

One additional benefit of using a neural network as opposed to learning the variational parameters directly is that the neural network can capture the temporal patterns in the data that can not be captured by the variational parameters on their own, as the unrestricted dynamics model is extremely flexible and can cope with a rather drastic evolution of attributes and interaction matrices. Since the neural network is being trained to predict the parameters for time t given the history up to time $t-1$, it is being encouraged to look for temporal patterns in the data.

In all our experiments we use the well known Adam optimizer (Kingma and Ba 2014) with a learning rate of 0.01 to train the inference network. A separate inference network is trained for all time steps (in other words, to make predictions for time t we train the inference network with all the observations up to time $t-1$). Note that all networks have exactly the same number of parameters. While training, the parameters of the neural network that is used to make predictions at time t are initialized with the parameters of trained network for time $t-1$.

DATASET	#NODES	#LINKS	#STEPS	#REL
ENRON	143	2,347	16	1
UCI	1,809	16,822	13	1
YELP	6,569	95,361	16	1
ML-10M	20,537	43,760	13	1
YAGO	10,623	201089	186	10
WIKI	12,554	669,934	232	24

Table 1: Dataset Description. Here, #Nodes is the number of nodes N, #Links is number of edges, #Steps is the number of snapshots T, and #Rel is the number of relations types R.

4 Experiments

4.1 Dataset Description

We use the UCI (Panzarasa, Opsahl, and Carley 2009), Enron (Klimt and Yang 2004), Yelp[3], ML-10M (Harper and Konstan 2015), WIKI (Leblay and Chekol 2018), and YAGO (Mahdisoltani, Biega, and Suchanek 2014) datasets in our experiments. Table 1 summarizes the datasets and additional information is provided in the supplementary material.

4.2 Link Forecasting

We consider two settings for link forecasting: single-step and multi-step link forecasting. In single-step link forecasting, we are given a dynamic network up to time t as a sequence of snapshots $[\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(t)}]$ and the task is to predict $\mathbf{A}^{(t+1)}$. In multi-step link forecasting, the aim is to predict the next k snapshots $\mathbf{A}^{(t+1)}, \mathbf{A}^{(t+2)}, \ldots, \mathbf{A}^{(t+k)}$.

For evaluating link forecasting in communication and rating networks, we use the well known *Area Under Curve*-score (AUC-score). See Appendix D in the supplementary

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

Datasets	AUC	node2vec[a]	DynamicTriad[b]	DynGEM[c]	DynAERNN[d]	DySAT[e]	BAS	NLSM
Enron	Micro	83.72 ± 0.7	80.26 ± 0.8	67.83 ± 0.6	72.02 ± 0.7	85.71 ± 0.3	76.88 ± 0.5	$\mathbf{87.05 \pm 0.3}$
	Macro	83.05 ± 1.2	78.98 ± 0.9	69.72 ± 1.3	72.01 ± 0.7	$\mathbf{86.60 \pm 0.2}$	77.62 ± 0.5	86.24 ± 0.4
UCI	Micro	79.99 ± 0.4	77.59 ± 0.6	77.49 ± 0.4	79.95 ± 0.4	81.03 ± 0.2	78.79 ± 0.5	$\mathbf{86.24 \pm 0.4}$
	Macro	80.49 ± 0.6	80.28 ± 0.5	79.82 ± 0.5	83.52 ± 0.4	85.81 ± 0.1	83.84 ± 0.4	$\mathbf{88.9 \pm 0.3}$
Yelp	Micro	67.86 ± 0.2	63.53 ± 0.3	66.02 ± 0.2	69.54 ± 0.2	70.15 ± 0.1	70.21 ± 0.1	$\mathbf{81.38 \pm 0.2}$
	Macro	65.34 ± 0.2	62.69 ± 0.3	65.94 ± 0.2	68.91 ± 0.2	69.87 ± 0.1	69.40 ± 0.1	$\mathbf{80.12 \pm 0.3}$
ML-10M	Micro	87.74 ± 0.2	88.71 ± 0.2	73.69 ± 1.2	87.73 ± 0.2	90.82 ± 0.3	84.10 ± 0.4	$\mathbf{92.21 \pm 0.4}$
	Macro	87.52 ± 0.3	88.43 ± 0.1	85.96 ± 0.3	89.47 ± 0.1	$\mathbf{93.68 \pm 0.1}$	84.32 ± 0.3	92.39 ± 0.3

Table 2: Single-step link forecasting results. Micro and Macro AUC in % averaged over 10 runs with standard deviation. NLSM beats previous state-of-the-art results in almost all settings. Reported performance scores for other methods were taken from (Sankar et al. 2020). References: [a](Grover and Leskovec 2016), [b](Zhou et al. 2018), [c](Goyal et al. 2018), [d](Goyal, Chhetri, and Canedo 2020), [e](Sankar et al. 2020)

material for details about evaluation metrics. We compare our performance against existing approaches for both static as well as dynamic network representation learning. While training and evaluating, links formed with nodes that are not observed in the training time-steps are removed.

For Temporal KGs, we use a windowed approach for training the model to reduce the computational requirement as these datasets have a large number of snapshots. We use a moving window of size m having snapshots $\mathbf{A}^{(t-m+1)}, \mathbf{A}^{(t-m+2)}, \ldots, \mathbf{A}^{(t)}$ and predict future snapshots. This is done for all $t = 2, \ldots, T$. The window is shifted by one step if $t > m$. While shifting, we keep the parameters of GRUs fixed but change the initial embeddings of nodes and interaction matrices by initializing them with one step evolved embeddings from their respective GRUs.

We evaluate Temporal KGs in the multi-step link forecasting setting and use well known metrics like Mean Reciprocal Rank (**MRR**), **Hits@3**, and **Hits@10**. We use both raw and filtered versions of these metrics as explained in (Bordes et al. 2013) to compare our model with existing methods. See Appendix D in the supplementary material for more details about the evaluation metrics.

In all our experiments, we fix the value of K to 64. This value allows significant flexibility in the model while maintaining computational tractability. Larger values can also be used without significantly affecting the results. Similarly, we chose $s_\theta = s_\psi = 0.1$ and $\sigma_\theta = \sigma_\psi = 10$ for all experiments. Our model is rather robust to the choice of hyperparameters and dataset specific tuning is, in general, not required as exemplified by the fact that we reuse the same values of hyperparameters across all of our experiments.

4.3 Results

Single Relation Link Forecasting: In this setting, we train the model using snapshots until time t and forecast the links occurring at time $t + 1$ for each $t = 1, \ldots, T - 1$. We initialize the model for forecasting the links at time $t + 1$ with the model trained at time t, and then retrain it. We use Micro-AUC and Macro-AUC metrics for evaluating the performance. Micro-AUC is calculated considering all the links across the time-steps and Macro-AUC is the average of AUC at each time-step. These experiments use communication and rating networks and the results are reported in Table 2. We can see that our model NLSM outperforms all other approaches based on the Micro-AUC scores. We can also observe that our model performs much better as compared to the static network embedding method node2vec. This is because static models do not consider temporal dynamics.

To demonstrate the utility of having GRUs, we created a baseline (BAS). BAS directly approximates ELBO($\mathbf{\Phi}$) as a function of variational parameters in (12) and (13). This model does not use a GRU for posterior inference. Instead, it learns embeddings for all nodes independently at each time-step. Then, for predicting the future links, it uses the latest embedding of the nodes. Note that BAS performs poorly across datasets. This shows that the regularization provided by the GRUs allows us to better capture the temporal patterns and improve the quality of inference.

Multi-Relational Link Forecasting: For multi-relational link forecasting, we use the temporal KGs YAGO and WIKI. We use the heterogeneous variant of our model NLSM as explained in 2.3. As before, we use a windowed training approach and with window size $m = 5$ as in (Jin et al. 2019). Testing is done by inferring the future embeddings via the node embedding GRUs. We followed the train-test split used by existing works (Jin et al. 2019). For YAGO we test on the last six time steps and for WIKI we test on the last 10 time steps. The performance is reported in Table 3. It can be observed that our model performs much better than the previous state-of-the-art model RE-NET (Jin et al. 2019) in all the settings.

See Appendix E in the supplementary material (Gracious et al. 2021) for additional details regarding our experimental setup. Appendix F presents additional results on smaller datasets and explores the effect of changing K. Appendix G presents a qualitative case study to demonstrate that learned embeddings may also offer interpretable insights about the network dynamics.

5 Related Work

Statistical Network Analysis: One of the first successful statistical model for dynamic networks was proposed in

Method	WIKI - filtered			WIKI - raw			YAGO - filtered			YAGO - raw		
	MRR	**H@3**	**H@10**	**MRR**	**H@3**	**H@10**	**MRR**	**H@3**	**H@10**	**MRR**	**H@3**	**H@1O**
Know-Evolve+MLP[a]	12.64	14.33	21.57	10.54	13.08	20.21	6.19	6.59	11.48	5.23	5.63	10.23
DyRep+MLP[b]	11.60	12.74	21.65	10.41	12.06	20.93	5.87	6.54	11.98	4.98	5.54	10.19
RE-NET[c]	53.57	54.10	55.72	32.44	35.42	43.16	66.80	67.23	69.77	48.60	54.20	63.59
NLSM	**56.70**	**57.80**	**61.10**	**35.25**	**38.60**	**47.55**	**69.40**	**71.25**	**73.90**	**52.50**	**59.20**	**68.40**

Table 3: Results on Temporal KGs. Reported scores are in % averaged over 5 runs. Our model performs better than existing approaches. The performance scores for other approaches have been taken from (Jin et al. 2019). References: [a](Trivedi et al. 2017), [b] (Trivedi et al. 2019), [c](Jin et al. 2019)

(Xing, Fu, and Song 2010). It is an extension of the well known Mixed Membership Stochastic Blockmodel (Airoldi et al. 2008) with the additional assumption that parameters evolve via a Gaussian random walk. Since then, multiple researchers have proposed extensions of static network models like Stochastic Blockmodel (Holland, Laskey, and Leinhardt 1983) to the case of dynamic networks (Yang et al. 2011; Xu and Hero 2014; Xu 2015).

Another class of models extend the general latent space model for static networks to the dynamic network setting (Sarkar and Moore 2005; Foulds et al. 2011; Heaukulani and Ghahramani 2013; Kim and Leskovec 2013; Sewell and Chen 2015, 2016; Gupta, Sharma, and Dukkipati 2019). Our proposed model also falls under this category. The basic idea behind such models is to represent each node by an embedding (which may change with time) and model the probability of an edge as a function of the embeddings of the two endpoints. All of these approaches (except (Gupta, Sharma, and Dukkipati 2019)) use an MCMC based inference procedure that does not directly support neural network based inference. However, unlike these previous approaches, in our model the role of each attribute in latent vector can also change. This is a rather distinctive feature of NLSM, allowing us to capture both local dynamics (the evolution of attributes in node latent vector) and global dynamics (the evolving role of attributes). In addition to that, our model for static network snapshots is fully differentiable which allows us to use a neural network based variational inference procedure as opposed to most existing methods that use MCMC based inference.

Dynamic Networks: For incorporating temporal properties into embedding learning, methods were devised to model the evolution of networks over time. One such work (Zhou et al. 2018) uses the triadic closure property where it tries to model the probability that an open triad in the current time-step will become closed in future. Recent focus of dynamic network representation learning is on adapting graph neural network based static representation learning methods (Hamilton, Ying, and Leskovec 2017; Petar et al. 2018; Zitnik, Agrawal, and Leskovec 2018) to the case of dynamic networks. For example, Goyal et al. (2018) use an incremental learning of graph auto-encoder at each time-step. The above mentioned approaches can only model short term dynamics in the networks. For capturing long-term dynamics, recurrent neural network and temporal attention is applied. In (Goyal, Chhetri, and Canedo 2020), graph encoders are used with recurrent neural network for modelling the evolution of node embeddings. In (Sankar et al. 2020), graph attention networks are used with dynamic self-attention for capturing long-term dynamics of nodes.

Temporal KGs: The model proposed in this paper is an extrapolation method for temporal KGs. Here, we forecast the future links by observing the dynamics in the temporal KG in the past. In an existing approach known as Know-Evolve (Trivedi et al. 2017), temporal point process is used for predicting links in a temporal KG. Here, the authors assume that the properties of entities evolve as they interact with other entities, and use these entity representations as an input to a bilinear scoring function defined by a relationship matrix. The score of the bilinear function is used as a conditional density in the temporal point process. This model is extended in DyRep (Trivedi et al. 2019), where a attention based neighborhood aggregator is added to the entity evolution model. In a recent work RE-NET (Jin et al. 2019), a graph convolution encoder was used to aggregate the neighborhood information at each time-step in the past. A recurrent neural network then aggregates the historical neighborhood information which is then used for forecasting the probability of links in the KG in future.

6 Conclusion

In this paper, we presented a new statistical model called Neural Latent Space Model (NLSM) for dynamic networks. Unlike most existing approaches which focus on modelling homogeneous dynamic networks, our approach seamlessly works for both homogeneous as well as heterogeneous dynamic networks. This is achieved by using relation specific interaction matrices for modelling links. For heterogeneous networks like the temporal KGs which have multiple type of relations, our model has an interaction matrix for each relation type. We also developed a neural network based variational inference procedure for performing inference in NLSM. Through our experiments, we demonstrated the utility of our approach by using it to perform link forecasting where we achieved state-of-the-art performance on several datasets in both single relation and multi-relational setting.

Acknowledgements

This work arose from a visit of Ambedkar Dukkipati to Eurandom, with the generous support of the STAR cluster and Eurandom. The authors TG, SG, AK and AD would like to thank the Ministry of Human Resource Development (MHRD), Government of India, for their generous funding towards this work through the UAY Project: IISc 001. SG is supported by WIPRO PhD Fellowship.

References

Airoldi, E. M.; Blei, D. M.; Fienberg, S. E.; and Xing, E. P. 2008. Mixed Membership Stochastic Blockmodels. *Journal of Machine Learning Research (JMLR)* 9: 1981–2014.

Blei, D. M.; Kucukelbir, A.; and McAuliffe, J. D. 2017. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association* 112(518): 859–877.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating Embeddings for Modeling Multi-relational Data. *Advances in Neural Information Processing Systems 26* 2787–2795.

Cho, K.; van Merrienboer, B.; Bahdanau, D.; and Bengio, Y. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8), 2014* .

Dettmers, T.; Minervini, P.; Stenetorp, P.; and Riedel, S. 2018. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Fortunato, S. 2010. Community Detection in Graphs. *Physics Reports* 486(3): 75 – 174.

Foulds, J.; DuBois, C.; Asuncion, A.; Butts, C.; and Smyth, P. 2011. A Dynamic Relational Infinite Feature Model for Longitudinal Social Networks. *Proceedings of Machine Learning Research (PMLR)* 15: 287–295.

Goldenberg, A.; Zheng, A. X.; Fienberg, S. E.; and Airoldi, E. M. 2010. A Survey of Statistical Network Models. *Foundations and Trends® in Machine Learning* 2(2): 129–233.

Goyal, P.; Chhetri, S. R.; and Canedo, A. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems* 187: 104816.

Goyal, P.; Kamra, N.; He, X.; and Liu, Y. 2018. DynGEM: Deep Embedding Method for Dynamic Graphs. *CoRR* abs/1805.11273.

Gracious, T.; Gupta, S.; Kanthali, A.; Castro, R. M.; and Dukkipati, A. 2021. Supplementary Material for Neural Latent Space Model for Dynamic Networks and Temporal Knowledge Graphs. *arXiv/1911.11455* .

Grover, A.; and Leskovec, J. 2016. Node2vec: Scalable Feature Learning for Networks. *Proceedings of International Conference on Knowledge Discovery and Data Mining* 855 – 864.

Gupta, S.; Sharma, G.; and Dukkipati, A. 2019. A Generative Model for Dynamic Networks with Applications. *In Proceedings of Thirty-Third AAAI Conference on Artificial Intelligence (AAAI)* .

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. *Advances in Neural Information Processing Systems 30* 1024–1034.

Harper, F. M.; and Konstan, J. A. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5(4).

Heaukulani, C.; and Ghahramani, Z. 2013. Dynamic Probabilistic Models for Latent Feature Propagation in Social Networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, volume 28, I–275–I–283.

Holland, P. W.; Laskey, K. B.; and Leinhardt, S. 1983. Stochastic Blockmodels: First steps. *Social Networks* 5(2): 109 – 137.

Jin, W.; Jiang, H.; Qu, M.; Chen, T.; lin Zhang, C.; Szekely, P. A.; and Ren, X. 2019. Recurrent Event Network : Global Structure Inference Over Temporal Knowledge Graph. *arXiv: Learning* .

Kim, M.; and Leskovec, J. 2012. Multiplicative Attribute Graph Model of Real-World Networks. *Internet Mathematics* 8(1-2): 113–160.

Kim, M.; and Leskovec, J. 2013. Nonparametric Multigroup Membership Model for Dynamic Networks. *Advances in Neural Information Processing Systems* 26: 1385–1393.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization. *Proceedings of 3rd International Conference on Learning Representations (ICLR)* .

Kingma, D. P.; and Welling, M. 2013. Auto-Encoding Variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations (ICLR)* .

Klimt, B.; and Yang, Y. 2004. Introducing the Enron Corpus. In *CEAS*.

Leblay, J.; and Chekol, M. W. 2018. Deriving Validity Time in Knowledge Graph. In *Companion Proceedings of the The Web Conference 2018*, 1771–1776.

Mahdisoltani, F.; Biega, J. A.; and Suchanek, F. M. 2014. YAGO3: A Knowledge Base from Multilingual Wikipedias. In *CIDR*.

Panzarasa, P.; Opsahl, T.; and Carley, K. M. 2009. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology* 60(5): 911–932.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 701–710.

Petar, V.; Guillem, C.; Arantxa, C.; Adriana, R.; Pietro, L.; and Yoshua, B. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Sankar, A.; Wu, Y.; Gou, L.; Zhang, W.; and Yang, H. 2020. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. *Proceedings of the 13th International Conference on Web Search and Data Mining* .

Sarkar, P.; and Moore, A. W. 2005. Dynamic Social Network Analysis using Latent Space Models. *SIGKDD Explorations* 7(2): 31–40.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective Classification in Network Data. *AI Magazine* 29(3): 93.

Sewell, D. K.; and Chen, Y. 2015. Latent Space Models for Dynamic Networks. *Journal of the American Statistical Association* 110(512): 1646–1657.

Sewell, D. K.; and Chen, Y. 2016. Latent Space Models for Dynamic Networks with Weighted Edges. *Social Networks* 44: 105 – 116.

Sun, Z.; Deng, Z.-H.; Nie, J.-Y.; and Tang, J. 2019. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *International Conference on Learning Representations*.

Trivedi, R.; Dai, H.; Wang, Y.; and Song, L. 2017. Know-Evolve: Deep Temporal Reasoning for Dynamic Knowledge Graphs. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 3462–3471.

Trivedi, R.; Farajtabar, M.; Biswal, P.; and Zha, H. 2019. DyRep: Learning Representations over Dynamic Graphs. In *International Conference on Learning Representations*.

Xing, E. P.; Fu, W.; and Song, L. 2010. A state-space mixed membership blockmodel for dynamic network tomography. *The Annals of Applied Statistics* 4(2): 535–566.

Xu, K. S. 2015. Stochastic Block Transition Models for Dynamic Networks. *Journal of Machine Learning Research (JMLR)* 38: 1079–1087.

Xu, K. S.; and Hero, A. O. 2014. Dynamic Stochastic Blockmodels for Time-Evolving Social Networks. *IEEE Journal of Selected Topics in Signal Processing* 8(4): 552–562.

Yang, T.; Chi, Y.; Zhu, S.; Gong, Y.; and Jin, R. 2011. Detecting Communities and their Evolutions in Dynamic Social Networks - A Bayesian approach. *Machine Learning* 82(2): 157–189.

Yilmaz, A.; Javed, O.; and Shah, M. 2006. Object Tracking: A Survey. *ACM Comput. Surv.* 38(4).

Zhou, L.; Yang, Y.; Ren, X.; Wu, F.; and Zhuang, Y. 2018. Dynamic network embedding by modeling triadic closure process. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Zitnik, M.; Agrawal, M.; and Leskovec, J. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* 34(13): i457–i466.