Transformer-Style Relational Reasoning with Dynamic Memory Updating for Temporal Network Modeling

Dongkuan Xu[1], **Junjie Liang**[1], **Wei Cheng**[2], **Hua Wei**[1], **Haifeng Chen**[2], **Xiang Zhang**[1]

[1]The Pennsylvania State University
{dux19, jul672, hzw77, xzz89}@psu.edu
[2]NEC Laboratories America, Inc.
{weicheng, haifeng}@nec-labs.com

Abstract

Network modeling aims to learn the latent representations of nodes such that the representations preserve both network structures and node attribute information. This problem is fundamental due to its prevalence in numerous domains. However, existing approaches either target the static networks or struggle to capture the complicated temporal dependency, while most real-world networks evolve over time and the success of network modeling hinges on the understanding of how entities are temporally connected. In this paper, we present TRRN, a transformer-style relational reasoning network with dynamic memory updating, to deal with the above challenges. TRRN employs multi-head self-attention to reason over a set of memories, which provides a multitude of shortcut paths for information to flow from past observations to the current latent representations. By utilizing the policy networks augmented with differentiable binary routers, TRRN estimates the possibility of each memory being activated and dynamically updates the memories at the time steps when they are most relevant. We evaluate TRRN with the tasks of node classification and link prediction on four real temporal network datasets. Experimental results demonstrate the consistent performance gains for TRRN over the leading competitors.

Introduction

Network modeling is a fundamental problem due to its prevalence in numerous domains, such as knowledge base, social media and bioinformatics (Wu et al. 2020). Network modeling aims to learn the latent representations of nodes, which preserve both the structure properties of nodes and the node attribute information. Such representations benefit various applications, such as node classification, link prediction and community detection (Wu et al. 2020; Zhang, Cui, and Zhu 2018; Zhou et al. 2018), etc. Over the years, numerous efforts are made to improve the performance of network modeling, such as DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), GCN (Kipf and Welling 2017), GAT (Veličković et al. 2018), GraphSAGE (Veličković et al. 2018), which model the static networks.

However, in many real-world applications, networks are often dynamic and may evolve over time. To model temporal network, a single (static) observation is not sufficient to learn the latent representations, such as using GNNs (Figure 1(a)). It makes more sense to reason over the history of past observations such that sufficient information can be extracted for current time step decision-making. For this problem, an intuitive choice is to apply the gated RNNs (Hochreiter and Schmidhuber 1997) (Figure 1(b)) which maintain the information from past observations via the recurrent state vectors. However, it would ignore the network structure properties. Some recent progresses have been made on jointly modeling the spatial and temporal contextual information (He et al. 2019; Xu et al. 2019a,b) (Figure 1(c)). Despite their success, their capacity for learning complicated temporal dependency is limited. To successfully model temporal network, it is desirable to understand how entities are connected temporally. For this problem, we argue that the relational reasoning ability plays an essential role (Battaglia et al. 2018), which helps comprise a capacity to compare and contrast information observed at different time steps.

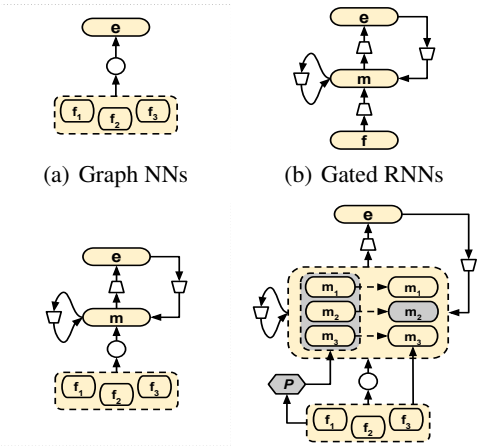


Figure 1: Four architectures for temporal network modeling. $\{\mathbf{f}_i\}$ represent different factors that influence node behaviors, such as network topology, attribute information of nodes and edges, etc. $\mathbf{e}$ is the latent representation of node to be learned. $\{\mathbf{m}_k\}$ are memories. Circles are aggregation layers. Trapeziums are gate layers. The hexagon with letter *p* inside is the policy network.

In this work, we focus on the problem of temporal network modeling. Given a sequence of network snapshots for the same set of nodes, where each node is coupled with attributes and has a unique class label over time. Both network topology and node attributes evolve over time. The task is to learn the latent representation for each node that considers temporal patterns of both topological structures and node attributes. We evaluate the learned representations on different downstream tasks, such as node classification and link prediction. The problem is challenging mainly for three reasons. First, capturing the complicated temporal dependency is hard. For temporal networks, the evolution distributes in different time steps. How to effectively compartmentalize and relate the information of different time steps is extremely interesting but at the same time quite challenging. Second, learning contextualized representations of nodes is also challenging. Node behaviors are often determined by various factors as denoted by $\{\mathbf{f}_i\}$ in Figure 1, such as node attributes and network topology structures, etc. Different factors at different time periods influence node representations diversely. How to embed them into node representations and differentiate their influence on node representations remains a challenging problem. Third, we highlight the sparse dynamics in the evolution of temporal networks. Dynamic systems in real-world are often characterized by independent and sparsely interacting dynamic processes (Goyal et al. 2019). It is desirable and challenging to capture the most relevant ones at time steps and update the dynamic processes sparsely (Bengio 2017).

In an effort to better model temporal networks, we propose TRRN, a transformer-style relational reasoning network with dynamic memory updating. Specifically, similar to memory-augmented architectures (Santoro et al. 2016, 2018), TRRN, as illustrated in Figure 1(d), adopts a set of memories to enhance temporal capacity. It applies Transformer-style self-attention to allow for interactions between memories over time, which helps compartmentalize and relate information of different time steps explicitly. To learn the contextualized representations of nodes, TRRN feeds different factors along with the updated memories into a gated recurrent network, from which the generated representations is able to consider the variety of contexts (factors) that the node involves. The influence of different factors are reflected in the attention matrix. Moreover, to model the sparse dynamics of networks, TRRN takes inspiration from conditional computation (Bengio, Léonard, and Courville 2013) and employs the policy networks to selectively activate and update the most relevant memories at each time step, on a per node basis. However, as the decisions determining which memories need to be activated are non-differentiable, we introduce differentiable binary routers based upon the Gumbel-softmax reparameterization (Jang, Gu, and Poole 2017) to train policy networks.

We validate the proposed model on node classification and link prediction tasks, using four real-world temporal network datasets. Experimental results show that TRRN outperforms the leading competitors by a large margin. We demonstrate the benefits of dynamically updating memories through an ablation study and visualize the activation decision vectors together with the attention weights to provide interpretability of results. We summarize our contributions as follows.

Method	Relational Bias	Relational Reasoning	Model Interp.	Different Factors	Sparse Dynamics
DeepWalk	S.	×	×	×	×
GAT	S.	×	×	∗	×
GCN	S.	×	×	∗	×
GraphSAGE	S.	×	×	∗	×
node2vec	S.	×	×	×	×
LSTM	T.	×	×	×	×
GRU	T.	×	×	×	×
DynGEM	S.+T.	×	×	×	×
DynAERNN.	S.+T.	×	×	×	×
DANE	S.+T.	×	×	∗	×
DySAT	S.+T.	∗	∗	∗	×
STAR	S.+T.	∗	∗	∗	×
Our Work	S.+T.	✓	✓	✓	✓

Table 1: A comparison of published approaches for temporal network modeling. S. and T. represent spatial relational bias and temporal relational bias respectively. Model Interp denotes model interpretability. ✓, ∗, × represent true, not exactly true and false, respectively.

- We analyze major challenges for temporal network modeling, including the complicated temporal dependency, the contextualized representations of nodes and the sparse dynamics of network evolution, which encourages us to find the complementarity between memory-augmented architectures and conditional computation.
- We extend the strength of multi-head self-attention via policy networks augmented with differential routers, proposing TRRN, in which complicated temporal dependency is learned, nodes in temporal networks can be contextualized flexibly and sparse dynamics of network evolution is captured.
- We summarize the existing network modeling approaches in a unified manner and conduct extensive experiments on four real datasets. The results show that TRRN outperforms the leading competitors in accuracy, and provides better interpretability as well.

Problem Definition

A temporal network, denoted by $\mathbb{G} = (\mathcal{G}^1, \mathcal{G}^2, \cdots, \mathcal{G}^T)$, is a collection of snapshots of a network at different time steps. $\mathcal{G}^t = (\mathcal{V}, \mathbf{A}^t, \mathbf{X}^t)$ is the snapshot at time step t. $\mathcal{V}$ is the set of nodes that is fixed for all time steps. Each node has a consistent label across T time steps. $\mathbf{A}^t \in \mathbb{R}^{N\times N}$ is the adjacency matrix. $\mathbf{X}^t \in \mathbb{R}^{N\times d}$ is the node attribute matrix. Unlike some previous methods that do not consider node attributes and assume links can only be added over time, we allow for node attributes and support the removal of links over time. The goal of temporal network modeling is to learn the latent representation $\mathbf{e}_v^t$ for every node $v \in \mathcal{V}$ at time step $t \in \{1,2,\cdots,T\}$, such that $\mathbf{e}_v^t$ preserves both network structures and node attributes related to node v.

Related Work

A comparison of the leading approaches that can be used to model temporal network is illustrated in Table 1.

Static Network Modeling Network modeling has drawn extensive research attention in recent years (Perozzi, Al-Rfou,

and Skiena 2014; Grover and Leskovec 2016; Wang, Cui, and Zhu 2016; Kipf and Welling 2017; Wang et al. 2017; Hamilton, Ying, and Leskovec 2017; Veličković et al. 2018). For example, a biased random walk is utilized to learn richer node representations by exploring diverse local information (Grover and Leskovec 2016). An attention is applied to generate node representations by learning the relationship between a node and its neighbors (Veličković et al. 2018).

Temporal Network Modeling However, many real-world networks are temporal. Some recent progresses have been made on modeling temporal networks (Zhu et al. 2018; Du et al. 2018; Goyal, Chhetri, and Canedo 2018; Ma et al. 2018; Zuo et al. 2018; Xu et al. 2019b; Sankar et al. 2020). An offline way to learn node representations in terms of network topology and node attributes is introduced (Li et al. 2017). A method based on autoencoder is proposed to generate the embedding of snapshot at time t from the embedding at time t-1 (Goyal et al. 2018). DySAT (Sankar et al. 2020) models network by appling self-attention along the two dimensions of structural neighborhood and temporal dynamics. STAR (Xu et al. 2019b) is a spatio-temporal attentive RNN, proposed to learn node representations in temporal networks. However, the capacity of these approaches for learning complicated temporal dependency is limited.

Memory-Based Architectures Many neural network approaches that successfully model sequential data utilize memory systems. However, the standard memory architectures like LSTM might struggle when tasks are involved in understanding how entities are related. Some memory-augmented architectures (Sukhbaatar et al. 2015; Santoro et al. 2016; Graves et al. 2016; Santoro et al. 2018; Goyal et al. 2019) are further proposed to address this issue. For example, RMC (Santoro et al. 2018) and RIMs (Goyal et al. 2019) apply self-attention on a set of memories similar to our TRRN. However, RMC updates all memories at each time and RIMs updates k memories. In contrast, TRRN selectively activates memories and update them dynamically. In addition, TRRN is proposed for network modeling, while these methods are proposed for reinforcement learning tasks.

The TRRN Model

The architecture of TRRN is shown in Figure 1(d). We first introduce how to determine the memories to be activated, followed by how to update the selected memories. Then we elaborate how to generate the contextualized representations, and last we give the learning objective.

Activating Memory Selectively

Figure 2 illustrates how to activate memories selectively, which allows each node to have its own memory activation policy at each time step. TRRN applies Gumbel Softmax sampling to jointly train the policy network and the target learning task. The output of policy network is sampled to produce activation decisions of which memories to be activated.

Extraction of Network Factors To generate contextualized representations, we mainly consider two factors, node attributes and network topology, which are believed to influence node representations most. Other factors like edge weights can be easily utilized by our method.

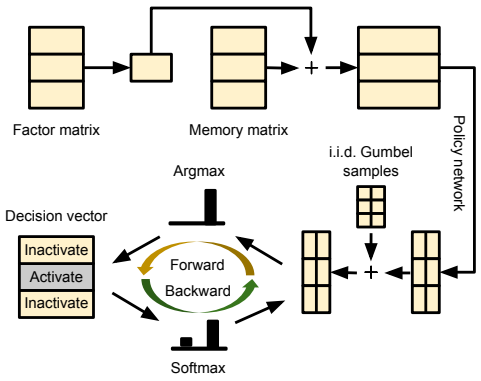


Figure 2: Illustration of activating memories selectively.

To extract node attribute factor, we feed node attributes into a fully-connected layer. The output vector is used as the attribute factor. How to extract network topology factor is flexible and we use random walk with restart (RWR) (Cao, Lu, and Xu 2016) in this work. Given a snapshot $\mathcal{G}^t = (\mathcal{V}, \mathbf{A}^t, \mathbf{X}^t)$ and a node v, the r-step RWR vector is defined as

$$\mathbf{p}^{(r)} = c\mathbf{p}^{(r-1)}[(\mathbf{D}^{-1})\mathbf{A}^t] + (1-c)\mathbf{p}^{(0)} \in \mathbb{R}_+^{1\times N}, \quad (1)$$

where $p_u^{(r)}$ represents the probability of node u transitions from v after r steps. $\mathbf{p}^{(0)}$ is the initial vector with $p_v^{(0)}$=1 and all other entries are 0. $\mathbf{D}$ is a diagonal matrix with $\mathbf{D}_{ii}$=$\sum_{j=1}^{N}\mathbf{A}_{ij}^t$. 1-$c$ is the probability that the random walker will restart from v. Thus, we use $\mathbf{a}_t = \sum_{r=1}^{R}\mathbf{p}^{(r)}$ to represent the topology information for node v at time step t, where R indicates the number of steps. After feeding $\mathbf{a}_t$ into a fully-connected layer, we use the output as the topology factor.

The Policy Network The policy network aims to estimate the probability of each memory being activated for each node. Its output is a matrix $\mathbf{B} \in \mathbb{R}^{n_M\times 2}$, of which rows correspond to n_M memories and columns represent two categories (activate and inactivate). The output $\mathbf{B}$ is further fed into the routers to produce the binary activation decisions.

At each time step, TRRN receives a memory matrix $\mathbf{M} \in \mathbb{R}^{n_M\times d_M}$ that stores the information from previous time steps and a factor matrix $\mathbf{F} \in \mathbb{R}^{n_F\times d_F}$ that represents an observation of n_F factors. TRRN concatenate each memory and the transformation of $\mathbf{F}$. The policy network further takes the concatenation as input. The form of policy networks is flexible and we opt to use a two-layer MLP as the policy network. Conceptually, it is defined as

$$\mathbf{B} = PolicyNetwork\left(\begin{bmatrix}\mathbf{m}_1^\top \oplus \tilde{\mathbf{f}}^\top \\ \mathbf{m}_2^\top \oplus \tilde{\mathbf{f}}^\top \\ \cdots \\ \mathbf{m}_{n_M}^\top \oplus \tilde{\mathbf{f}}^\top\end{bmatrix}\right) \in \mathbb{R}^{n_M\times 2}, \quad (2)$$

where $\oplus$ is the concatenation operator, $\mathbf{m}_k \in \mathbb{R}^{d_M}$ is the k-th memory, and $\tilde{\mathbf{f}} \in \mathbb{R}^{\tilde{d}_F}$ is generated by first flattening $\mathbf{F}$ and

then applying a transformation matrix $\boldsymbol{W} \in \mathbb{R}^{n_F d_F \times \tilde{d}_F}$. The policy network is jointly trained with other parts of TRRN. Its simple architecture makes the estimation of the probability matrix fast and efficient, which is similar to the design of the policy network in (Guo et al. 2019) and the design of Decision-Learner in (Ahmed and Torresani 2019).

Differentiable Binary Routers Given the output $\mathbf{B} \in \mathbb{R}^{n_M \times 2}$, a router is designed to produce the binary activation decisions for each memory. An intuitive way to achieve this goal is to select the position with maximum value of $\mathbf{b}_k = \{b_k^0, b_k^1\}$, where $k = \{1,2,\cdots,n_M\}$. However, this approach is non-differentiable. In this work, we adopt the Gumbel-Softmax sampling approach (Jang, Gu, and Poole 2017; Maddison, Mnih, and Teh 2017) that allows us to propagate gradients through the discrete nodes.

Specifically, we have two categories (activate and inactivate). The outputs $\{b_k^0, b_k^1\}$ are considered as the log probabilities $\{\log(p_k^0), \log(p_k^1)\}$ of the two categories for the k-th memory. Based on Gumbel-Max trick (Maddison, Mnih, and Teh 2017), we can draw samples from a Bernoulli distribution parameterized by class probabilities $\{p_k^0, p_k^1\}$ in the following way: we first draw i.i.d samples $\{g_k^0, g_k^1\}$ from a Gumbel distribution described by:

$$g_k^i = -\log(-\log(x)) \sim Gumbel, i \in \{0,1\}, \quad (3)$$

where $x \sim$ Uniform(0, 1). Then produce the discrete sample by adding g_k^i to introduce stochasticity:

$$z_k = \arg\max_i [b_k^i + g_k^i], i \in \{0,1\}. \quad (4)$$

Thus, we get a binary decision vector $\mathbf{z} = (z_1, z_2, z_{n_M})^\top \in \mathbb{R}^{n_M}$, in which the values indicate the memory is activated (z_k=1) or frozen (z_k=0).

However, the $\arg\max$ operation is non-differentiable. We can use the softmax as a continuous differentiable approximation to it, and generate a two-dimensional vector $\boldsymbol{\beta}_k$:

$$\beta_k^i = \frac{\exp\left((b_k^i + g_k^i)/\tau\right)}{\sum_{\hat{i}=0}^{1} \exp\left((b_k^{\hat{i}} + g_k^{\hat{i}})/\tau\right)}, i \in \{0,1\}, \quad (5)$$

where τ is the temperature to control the discreteness. Thus, we use the $\arg\max$ to make the binary activation decision on the forward pass, while approximate it with softmax on the backward pass, which is called the straight-through estimator (Jang, Gu, and Poole 2017).

Updating Memory Dynamically

Transformer-Style Self-Attention Transformer-style dot product self-attention (Vaswani et al. 2017) operates on a set of typed interchangeable objects and aims to relate different objects in order to compute representations of the same set. It has been shown to be useful in various task (Vaswani et al. 2017). Specifically, it applies a scaled dot-product of a query matrix $\mathbf{Q}$ to a key matrix $\mathbf{K}$, of which the result is normalized via a softmax to produce a set of weights. The resulting representations are the weighted average of a value matrix $\mathbf{V}$ based on the produced weights, which are computed as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}, \quad (6)$$

where the softmax is applied to each row of its argument matrix and d is the size of a key vector used as a scaling factor. Notably, the size d can be split into multiple heads and each head operates independently.

Intuitively, we can utilize transformer-style self-attention (generating $\mathbf{Q}$/$\mathbf{K}$/$\mathbf{V}$ via applying linear projections to $\mathbf{M}$) to allow memories to interact and update their content based upon the attended information. However, it is not capable to update the activated memories without ignoring the information from various factors that influence node behaviors.

Updating Activated Memories We propose to update the memories as follows.

$$\mathbf{Q} = (\mathbf{M} * \mathbf{z})\, \boldsymbol{W}^q, \quad (7)$$

$$\mathbf{K} = \left[\mathbf{M}; \mathbf{f}_1^\top; \mathbf{f}_2^\top; \cdots; \mathbf{f}_{n_F}^\top\right] \boldsymbol{W}^s, \quad (8)$$

$$\mathbf{V} = \left[\mathbf{M}; \mathbf{f}_1^\top; \mathbf{f}_2^\top; \cdots; \mathbf{f}_{n_F}^\top\right] \boldsymbol{W}^v, \quad (9)$$

$$\tilde{\mathbf{M}} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_s}}\right)\mathbf{V} + \mathbf{M} * (\mathbf{1} - \mathbf{z}) + \mathbf{M}, \quad (10)$$

where $\theta = (\boldsymbol{W}^q, \boldsymbol{W}^s, \boldsymbol{W}^v, \theta_z)$. $\mathbf{z}$ is the binary decision vector. $\left[\mathbf{M}; \mathbf{f}_1^\top; \mathbf{f}_2^\top; \mathbf{f}_{n_F}^\top\right]$ denotes the row-wise concatenation. $\mathbf{f_j}$ is the observation of the j-th factor.

In particular, to only update the activated memories, we use $\mathbf{z} \in \mathbb{R}^{n_M}$ to mask inactivated memories as shown in Equation (7), where $\mathbf{M} * \mathbf{z}$ is defined by

$$\mathbf{M} * \mathbf{z} = (\mathbf{m}_1 z_1, \cdots, \mathbf{m}_{n_M} z_{n_M})^\top. \quad (11)$$

In Equations (8)-(9), to consider the influence from different factors, we incorporate the vector representations of factors into new memory matrix .

In Equation (10), softmax $\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_s}}\right)\mathbf{V}$ represents the updated activated memories, $\mathbf{M} * (\mathbf{1} - \mathbf{z})$ represents the kept inactivated memories, and $\mathbf{M}$ is a residual connection.

Notably, we do not mask the inactivated memories when generating the key and value matrices (Equations (8)-(9)). This is because we allow the activated memories to read from all memories. The intuition is that inactivated memories might still store contextual information related to the activated memories, though they are not related to the current input factors.

Learning Contextualized Representations

Different factors at different time periods influence node representations diversely. To consider the observations of different factors at each time step, we introduce the following recurrence to generate the contextualized representations. The recurrence of each memory is independent, but shares the same set of parameters with others. Implementing recurrence is flexible and we opt to embed it into an LSTM.

$$\begin{bmatrix} \mathbf{g}_k^t \\ \mathbf{i}_k^t \\ \mathbf{o}_k^t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \end{bmatrix} \{\boldsymbol{W}[\mathbf{e}_k^{t-1} \oplus \mathbf{f}_1^t \oplus \mathbf{f}_2^t \oplus \mathbf{f}_{n_F}^t] + \boldsymbol{b}\}, \quad (12)$$

$$\mathbf{m}_k^t = \mathbf{g}_k^t \odot \mathbf{m}_k^{t-1} + \mathbf{i}_k^t \odot \psi(\tilde{\mathbf{m}}_k^t), \quad (13)$$

$$\mathbf{e}_k^t = \mathbf{o}_k^t \odot \tanh(\mathbf{m}_k^t), \quad (14)$$

$$\mathbf{e}^t = [\mathbf{e}_1^t \oplus \mathbf{e}_2^t \oplus \mathbf{e}_{n_M}^t], \quad (15)$$

where $\mathbf{e}^t \in \mathbb{R}^{n_M d_M}$ is the contextualized representation of a node and $\mathbf{e}_k^t \in \mathbb{R}^{d_M}$ is generated from the k-th memory. $\odot$ denotes the element-wise multiplication operator.

Equation (12) describes how to generate the forget, input and output gates for the k-th memory, which considers the previous node representation and all the factor observations. Equation (13) describes how to generate a new memory, where the updated memory $\tilde{\mathbf{m}}_k^t$ is used as a candidate. ψ is a transformation function and we use a two-layer MLP with layer normalization. Equation (14) describes how to generate a part of contextualized representation from the k-th memory. Equation (15) concatenates the representations from all memories as the contextualized representation of a node.

Objective Functions

Two Tasks Given the node representations $\{\mathbf{e}_v^1, \mathbf{e}_v^2, \cdots, \mathbf{e}_v^T\}$ for node $v \in \mathcal{V}$, we consider tasks of node classification and link prediction to evaluate the representation quality. The two tasks share a similar objective function:

$$J = L_{ce} + \lambda P_{nn}, \tag{16}$$

where $L_{ce} = -\frac{1}{N} \sum \mathbf{y} \log(\tilde{\mathbf{y}})$ is the cross-entropy loss, P_{nn} is the penalization term for parameters to address over-fitting, and λ is a hyper-parameter.

For node classification, $\mathbf{y}$ is the class label. $\tilde{\mathbf{y}}$ is the estimate, which is produced by $\tilde{\mathbf{y}} = \text{softmax}(\boldsymbol{W}_o \mathbf{e}_v^T + \boldsymbol{b}_o)$. $\boldsymbol{W}_o \in \mathbb{R}^{c \times n_M d_M}$ and $\boldsymbol{b}_o \in \mathbb{R}^c$ are parameters. c is the number of classes. For link prediction, $\mathbf{y}$ is the binary label, indicating whether there is a link between two nodes. $\tilde{\mathbf{y}}$ is the estimate, which is produced by a logistic regression classifier, i.e., $\tilde{\mathbf{y}} = \{h(\mathbf{e}_v^t, \mathbf{e}_u^t), 1 - h(\mathbf{e}_v^t, \mathbf{e}_u^t)\}$, where $h(\mathbf{e}_v^t, \mathbf{e}_u^t) = \sigma((\boldsymbol{W}_o \mathbf{e}_v^t)^\top (\boldsymbol{W}_o \mathbf{e}_u^t))$ and $\boldsymbol{W}_o \in \mathbb{R}^{d_o \times n_M d_M}$ is a parameter.

Sparsity Constrains We add the sparsity constraint on the binary activation decision vector $\mathbf{z} = (z_1, z_2, z_{n_M})^\top \in \mathbb{R}^{n_M}$ of each time step. We want TRRN to avoid activating every memory and capture the most relevant ones at each time step, which reflects the sparse dynamics in the evolution of temporal networks. Inspired by (Ke et al. 2018), we add a penalty term $\lambda_s \sum_{t=1}^T S(\mathbf{z}^t)$ into the objective function. $S(\mathbf{z}^t)$ represents the penalty that TRRN needs to pay for activating memories at time step t. The penalty term is defined as

$$\lambda_s \sum_{t=1}^{T} S(\mathbf{z}^t) = \lambda_s \sum_{t=1}^{T} \text{ReLU}((\sum_{k=1}^{n_M} z_k^t) - \gamma n_M), \tag{17}$$

where λ_s and γ are hyper-parameters. n_M is the number of memories. γ is the proportion of memories that are without being penalized when they are activated. Intuitively, each activated memory above the threshold γn_M is penalized.

Complexity Analysis TRRN is local in time and the input sequence length does not affect its storage requirements. The complexity per parameter is $\mathcal{O}(1)$ for each time step. Thus, the complexity of TRRN per time step is proportional to the number of parameters. The parameters of TRRN are from the policy network, memory updating, contextualized representation learning and the task layers. Based on the four parts, the complexity of TRRN is $\mathcal{O}(d_F d_M + c d_M n_M + n_F d_F^2 + n_M d_M^2)$ for node classification or $\mathcal{O}(d_F d_M + d_M d_o n_M + n_F d_F^2 + n_M d_M^2)$ for link prediction. Because $\mathcal{O}(d_M)=\mathcal{O}(d_F)$ and $\mathcal{O}(n_M)=\mathcal{O}(n_F)$, the complexity of per time step of TRRN is derived to $\mathcal{O}(c d_M n_M + n_M d_M^2)$ or $\mathcal{O}(d_M d_o n_M + n_M d_M^2)$.

Statistics	Task-fMRI	DBLP5	Epinions	Reddit
# Nodes	5000	6606	16025	8291
# Edges	1955488	42815	1144258	264050
# Attributes	20	100	20	20
# Time Steps	12	10	11	10
# Node Categories	10	5	10	4

Table 2: Dataset description.

Experiments

To analyze the node representation quality from different perspectives, we propose three research questions:

(RQ1) How does TRRN compare with the leading network modeling methods on node classification task?

(RQ2) How does TRRN compare with the competitors on link prediction task?

(RQ3) Are the proposed memory activating and updating mechanism in TRRN effective and interpretable?

Experimental Setup

Datasets We use four real temporal network datasets as shown in Table 2. Task-fMRI is a brain temporal network dataset, where nodes represent tidy cubes of brain tissue and are categorized into ten groups. Two nodes are connected if they show similar degree of activation during a time period (Gonzalez-Castillo et al. 2015). This dataset is extracted from the task based functional magnetic resonance imaging (fMRI) data[1], which is collected when the subject conducts different tasks successively. We apply PCA to the fMRI data of a time period to generate the node attributes of a network snapshot. DBLP5 is a co-author temporal network dataset, where nodes represent authors. The dataset is extracted from the bibliography website DBLP. The node attributes in a network snapshot are extracted from the titles and abstracts of the corresponding author's publications during a time period by word2vec (Mikolov et al. 2013). The authors in DBLP5 are from five areas. Epinions is a who-trust-whom temporal network dataset which is extracted from the product review website Epinions.com, where users decide whether to trust others to seek advice. Nodes represent users and two nodes are connected if one of them seeks advice from the other. Node attributes are generated from the reviews by word2vec. We select ten categories of products to construct the dataset. Reddit is a post temporal network dataset (reddit.com), where nodes represent posts. Two nodes are connected if their corresponding posts contain similar keywords. We apply the word2vec approach to the comments of a post to generate its node attributes. We select four categories of the posts.

Baselines We compare TRRN with some competitive baselines. DeepWalk (Perozzi, Al-Rfou, and Skiena 2014),

[1] https://tinyurl.com/y4hhw8ro

Method	Task-fMRI		DBLP5		Epinions		Reddit	
	ACC	AUC	ACC	AUC	ACC	AUC	ACC	AUC
DeepWalk	71.4±1.3	97.2±1.0	35.4±1.2	61.0±1.8	30.1±1.6	68.4±1.8	47.5±1.7	71.9±2.4
GAT	43.8±2.5	86.2±3.4	32.5±2.4	48.6±2.9	22.5±1.5	63.1±1.8	29.6±1.9	52.4±2.6
GCN	65.0±1.4	86.7±0.9	33.7±1.3	50.0±1.2	20.9±0.7	62.4±1.4	27.7±0.8	54.0±1.0
GraphSAGE	69.4±2.6	96.7±2.1	71.0±1.1	90.7±1.9	24.5±2.9	63.9±2.0	42.5±2.1	66.8±2.5
node2vec	71.0±1.5	96.8±1.8	36.9±1.1	64.2±1.1	32.8±1.5	70.2±1.6	48.0±1.3	72.2±1.1
LSTM	83.6±1.8	98.6±1.5	74.1±0.6	91.4±0.8	17.9±1.0	61.5±1.2	40.2±1.4	66.5±1.6
GRU	80.4±1.7	98.2±1.7	75.6±1.0	91.5±1.1	17.3±1.1	61.7±1.6	42.1±1.4	67.2±1.7
DynGEM	71.0±2.7	97.2±2.7	52.3±3.2	59.0±3.4	31.6±2.4	54.6±2.5	39.9±3.5	66.2±2.8
DANE	85.2±1.3	94.8±2.9	82.5±1.7	92.3±1.0	31.8±1.8	67.1±1.8	45.7±1.9	70.0±1.6
STAR	89.2±1.2	99.2±0.8	80.3±1.5	95.5±0.7	32.6±1.6	67.4±1.5	50.8±1.3	75.0±1.7
TRRN	**91.5±1.0**	**99.8±0.8**	**88.9±1.5**	**97.6±1.8**	**34.6±1.3**	**72.8±1.2**	**52.0±1.7**	**79.2±1.8**

Table 3: Node classification results (%).

GAT (Veličković et al. 2018), GCN (Kipf and Welling 2017), GraphSAGE (Veličković et al. 2018) and node2vec (Grover and Leskovec 2016) are well-known network modeling methods proposed for static networks. They are good at extracting spatial structure information, but ignore the temporal dependency of temporal networks. LSTM and GRU (Cho et al. 2014) are proposed to model temporal dependency via recurrent states and famous for their gate mechanism to address the vanishing (and exploding) gradient issues. DANE (Li et al. 2017), DynGEM (Goyal et al. 2018), DynAERNN (Goyal, Chhetri, and Canedo 2018) and STAR (Xu et al. 2019b) are temporal network modeling methods. DANE and STAR are capable of considering both network topology and node attributes. To gain insights about TRRN, we study some of its variants. TRRN-All is the variant that updates all memories at each time step. TRRN-Fix is the variant that updates a fixed number of memories (half of the total) at each time step. TRRN-S does not apply the sparsity constraint. More details of these methods are summarized in Table 1.

Other Settings In our experiments, λ and λ_s are set to the same, 10^{-3}. They are determined by grid-search from $\{0, 2\times10^{-4}, 5\times10^{-4}, 1\times10^{-3}, 2\times10^{-3}\}$. γ is set to 0.5, which is from $\{0.25, 0.5, 0.75\}$. R and n_F are set to 4 and 2 respectively. n_M is set to 4, which is from $\{2,3,4,5\}$. d_M, d_F and $\tilde{d}_F$ are set to the same, 20. The sizes of query, key, value are set to same, d_s=20 (we use three attention heads). They are determined from $\{10, 20, 30, 40\}$. We randomly select 4/5 of all training examples as the training set and 1/5 as the test set. We randomly select 1/10 of the training set as the validation set to determine the best hyper-parameters. 10-fold cross-validation is applied. TRRN is implemented with PyTorch and optimized by Adam (Kingma and Ba 2014)[2].

Node Classification Comparison (RQ1)

In this section, we conduct experiments on node classification. All models are trained on $\mathbb{G} = (\mathcal{G}^1, \mathcal{G}^2, \cdots, \mathcal{G}^T)$ to learn the node representations $\{\mathbf{e}_v^1, \mathbf{e}_v^2, \cdots, \mathbf{e}_v^T\}$ for $v \in \mathcal{V}$. Given these representations and the labels of a subset of nodes $\mathcal{V}_L$, node classification aims to classify the nodes in subset $\mathcal{V}_U$ whose labels are unknown, where $\mathcal{V} = \mathcal{V}_L \cup \mathcal{V}_U$. For the static network modeling methods, we first apply them to each network snapshot to generate the node representation at each time step. Then we concatenate the representations of all time steps into a vector. A fully-connected layer with softmax is applied to the vector to predict the node label.

Table 3 shows the node classification results. We observe that TRRN achieves consistent gains compared to the baselines across all datasets. LSTM and GRU perform well on Task-fMRI and DBLP, but show low performance on Epinions and Reddit. This is because the node representations of Task-fMRI and DBLP are dominated by node attributes, but they are dominated by network topology in Epinions and Reddit. DeepWalk and node2vec show high performance on Epinions and Reddit because of their advanatges of extracting topology information. STAR DANE and DynGEM show high performance in general, which is mainly because of their ability of utilizing both node attribute and network topology information. We conjecture that allowing relational reasoning on a set of memories helps TRRN capture the evolution of temporal networks and thus is responsible for achieving the superior performance.

Link Prediction Comparison (RQ2)

Link prediction is another widely used task to evaluate the quality of learned node representations to predict the temporal evolution of network structures (Wu et al. 2020). The training examples (node pairs) are created for time step t by sampling the links in $\mathcal{G}^t$ and an equal number of randomly sampled non-links. We split the training examples of each time step into training set and test set. All models are trained on training sets and tested on test sets. The node representations at time step t-1 are used to predict the links at time step t. i.e., classifying a node pair in test set into links and non-links. We evaluate the models at time step t for $t \in \{2,3,\cdots,T\}$.

Table 4 shows the link prediction results. TRRN shows the consistently superior performance. Notably, GAT shows comparable performance compared to STAR. One possible reason is that GAT has a good ability to extract local network structure information, which plays an important role on link prediction. TRRN outperforms STAR, which indicates the benefit of reasoning over a set of memories that provides multiple paths for information to flow from past to current. In addition, we compare the prediction results at each time step as shown in Figure 3. It is observed that TRRN shows more stable performance compared to GAT. This is because the static network modeling methods ignore the temporal

[2]Data and codes can be found in the authors' website.

Method	Task-fMRI		DBLP5	
	ACC	AUC	ACC	AUC
GAT	72.2±1.5	75.6±0.9	63.9±2.0	69.0±1.7
LSTM	68.1±2.7	71.3±0.7	59.7±1.4	65.9±1.7
STAR	71.9±0.9	73.4±1.7	63.9±1.5	67.4±1.5
TRRN	**75.3±1.9**	**78.0±0.9**	**66.5±0.7**	**70.9±1.4**

Table 4: Link prediction results (%).

Method	Task-fMRI		DBLP5	
	ACC	AUC	ACC	AUC
TRRN-All	88.7±1.9	97.6±1.1	86.0±1.8	94.3±1.6
TRRN-Fix.	90.1±2.3	98.4±1.6	85.2±1.9	95.2±1.8
TRRN-S	90.4±1.2	97.7±0.9	86.2±1.5	95.2±1.6
TRRN	**91.5±1.0**	**99.8±0.8**	**88.9±1.5**	**97.6±1.8**

Table 5: Ablation study on dynamic memory updating.

dependency of network evolution.

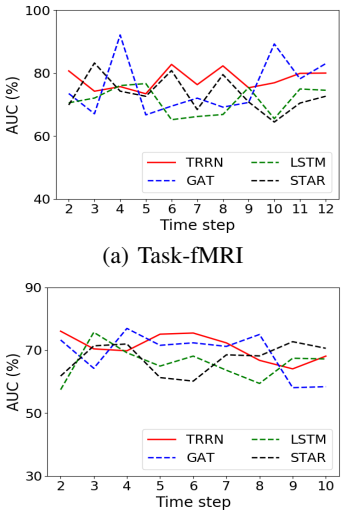
Analysis of Memory Activating & Updating (RQ3)

Since memory activating and updating play critical role in TRRN, we conduct exploratory analysis of: (effectiveness) how they influence the performance of TRRN, and (interpretability) whether the activation decisions are interpretable.

Effectiveness We conduct an ablation study by comparing TRRN with its variants TRRN-All, TRRN-Fix and TRRN-S. TRRN-All updates all memories by setting the activation decision vector $\mathbf{z}$ to $\mathbf{1}$. TRRN-Fix updates half of the memories. It feeds the concatenation of input factors and memories to an attention layer and the memories with the top half attention values are selected. TRRN-S removes the sparsity constrain. Table 5 shows the results (node classification). TRRN outperforms TRRN-All and TRRN-S, indicating the importance of selective memory activation and sparsity constrain on modeling network evolution. By comparing TRRN with TRRN-Fix, we see the benefit of updating memories dynamically.

Interpretability We visualize the (averaged) activation decision vectors. Figure 4(a) shows the results of two categories of nodes from Task-fMRI. It is observed the two categories of nodes activate different memories. However, they share more decision vector patterns at the first and last few periods. This is because both of the categories of nodes are active when the subject conduct the tasks during these periods.

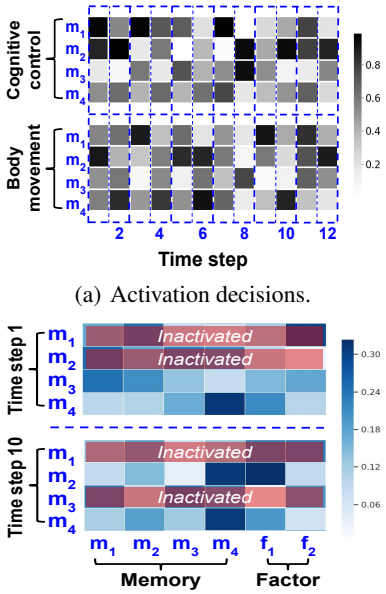
We also visualize the self-attention weight matrix, i.e., softmax $(\mathbf{Q}\mathbf{K}^\top/\sqrt{d_s})$ in Eq. (10). Figure 4(b) shows the results. We observe memories are activated dynamically in different periods.Moreover, node attribute factor (f_1) shows higher attention values (on activated memories) compared to network topology (f_2), which indirectly verifies attribute information is dominant in DBLP5. Thus, TRRN is able to provide effective and more importantly, interpretable evolution modeling results.



(a) Task-fMRI

(b) DBLP5

Figure 3: Link prediction results at each time step.



(a) Activation decisions.

(b) Self-attention matrices.

Figure 4: (a) The visualization of (averaged) activation decision vectors of two categories of nodes from Task-fMRI. (b) The visualization of self-attention weight matrices of a node from DBLP5 at two time steps .

Conclusion

In this paper, we propose a method TRRN to model temporal networks. To capture the complicated temporal dependency, TRRN employs transformer-style self-attention to reason over a set of memories. With the policy network augmented with differentiable routers, TRRN updates memories dynamically at the time steps where they are more relevant. Contextualized node representations are generated by considering both updated memories and different factors that influence node behaviors. Experimental results show TRRN outperforms the competitive baselines and provides interpretability for the modeling results.

Acknowledgements

This project was partially supported by NSF projects IIS-1707548 and CBET-1638320.

References

Ahmed, K.; and Torresani, L. 2019. STAR-Caps: Capsule networks with straight-through attentive routing. In *NeurIPS*, 9098–9107.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* .

Bengio, Y. 2017. The consciousness prior. *arXiv:1709.08568* .

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432* .

Cao, S.; Lu, W.; and Xu, Q. 2016. Deep neural networks for learning graph representations. In *AAAI*.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*.

Du, L.; Wang, Y.; Song, G.; Lu, Z.; and Wang, J. 2018. Dynamic Network Embedding: An Extended Approach for Skip-gram based Network Embedding. In *IJCAI*, 2086–2092.

Gonzalez-Castillo, J.; Hoy, C. W.; Handwerker, D. A.; Robinson, M. E.; Buchanan, L. C.; Saad, Z. S.; and Bandettini, P. A. 2015. Tracking ongoing cognition in individuals using brief, whole-brain functional connectivity patterns. *PNAS* 112(28): 8762–8767.

Goyal, A.; Lamb, A.; Hoffmann, J.; Sodhani, S.; Levine, S.; Bengio, Y.; and Schölkopf, B. 2019. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893* .

Goyal, P.; Chhetri, S. R.; and Canedo, A. 2018. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *arXiv:1809.02657* .

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

Graves, A.; Wayne, G.; Reynolds, M.; Harley, T.; Danihelka, I.; Grabska-Barwińska, A.; Colmenarejo, S. G.; Grefenstette, E.; Ramalho, T.; Agapiou, J.; et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538(7626): 471–476.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*, 855–864. ACM.

Guo, Y.; Shi, H.; Kumar, A.; Grauman, K.; Rosing, T.; and Feris, R. 2019. SpotTune: transfer learning through adaptive fine-tuning. In *CVPR*, 4805–4814.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NeurIPS*, 1024–1034.

He, B.; Zhou, D.; Xiao, J.; Liu, Q.; Yuan, N. J.; Xu, T.; et al. 2019. Integrating graph contextualized knowledge into pre-trained language models. *arXiv preprint arXiv:1912.00147* .

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8): 1735–1780.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical reparameterization with gumbel-softmax. In *ICLR*.

Ke, N. R.; Zolna, K.; Sordoni, A.; Lin, Z.; Trischler, A.; Bengio, Y.; Pineau, J.; Charlin, L.; and Pal, C. 2018. Focused hierarchical rnns for conditional sequence processing. In *ICML*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv:1412.6980* .

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Li, J.; Dani, H.; Hu, X.; Tang, J.; Chang, Y.; and Liu, H. 2017. Attributed network embedding for learning in a dynamic environment. In *CIKM*, 387–396. ACM.

Ma, Y.; Guo, Z.; Ren, Z.; Zhao, E.; Tang, J.; and Yin, D. 2018. Streaming Graph Neural Networks. *arXiv:1810.10627* .

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2017. The concrete distribution: A continuous relaxation of discrete random variables. *ICLR* .

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv:1301.3781* .

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*, 701–710. ACM.

Sankar, A.; Wu, Y.; Gou, L.; Zhang, W.; and Yang, H. 2020. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. In *WSDM*, 519–527.

Santoro, A.; Bartunov, S.; Botvinick, M.; Wierstra, D.; and Lillicrap, T. 2016. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, 1842–1850.

Santoro, A.; Faulkner, R.; Raposo, D.; Rae, J.; Chrzanowski, M.; Weber, T.; Wierstra, D.; Vinyals, O.; Pascanu, R.; and Lillicrap, T. 2018. Relational recurrent neural networks. In *NeurIPS*, 7299–7310.

Sukhbaatar, S.; Weston, J.; Fergus, R.; et al. 2015. End-to-end memory networks. In *NeurIPS*, 2440–2448.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NeurIPS*, 5998–6008.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *ICLR*.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *SIGKDD*, 1225–1234. ACM.

Wang, X.; Cui, P.; Wang, J.; Pei, J.; Zhu, W.; and Yang, S. 2017. Community preserving network embedding. In *AAAI*.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* .

Xu, D.; Cheng, W.; Luo, D.; Gu, Y.; Liu, X.; Ni, J.; Zong, B.; Chen, H.; and Zhang, X. 2019a. Adaptive neural network for node classification in dynamic networks. In *ICDM*, 1402–1407. IEEE.

Xu, D.; Cheng, W.; Luo, D.; Liu, X.; and Zhang, X. 2019b. Spatio-Temporal Attentive RNN for Node Classification in Temporal Attributed Graphs. In *IJCAI*, 3947–3953.

Zhang, Z.; Cui, P.; and Zhu, W. 2018. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202* .

Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* .

Zhu, D.; Cui, P.; Zhang, Z.; Pei, J.; and Zhu, W. 2018. High-order proximity preserved embedding for dynamic networks. *TKDE* 30(11): 2134–2144.

Zuo, Y.; Liu, G.; Lin, H.; Guo, J.; Hu, X.; and Wu, J. 2018. Embedding Temporal Network via Neighborhood Formation. In *SIGKDD*, 2857–2866. ACM.