arXiv:2006.10637v1 [cs.LG] 18 Jun 2020

Temporal Graph Networks for Deep Learning on Dynamic Graphs

Emanuele Rossi
Twitter
erossi@twitter.com

Ben Chamberlain
Twitter
bchamberlain@twitter.com

Fabrizio Frasca
Twitter
ffrasca@twitter.com

Davide Eynard
Twitter
deynard@twitter.com

Federico Monti
Twitter
fmonti@twitter.com

Michael Bronstein
Twitter
mbronstein@twitter.com

Abstract

Graph Neural Networks (GNNs) have recently become increasingly popular due to their ability to learn complex systems of relations or interactions arising in a broad spectrum of problems ranging from biology and particle physics to social networks and recommendation systems. Despite the plethora of different models for deep learning on graphs, few approaches have been proposed thus far for dealing with graphs that present some sort of dynamic nature (e.g. evolving features or connectivity over time). In this paper, we present Temporal Graph Networks (TGNs), a generic, efficient framework for deep learning on dynamic graphs represented as sequences of timed events. Thanks to a novel combination of memory modules and graph-based operators, TGNs are able to significantly outperform previous approaches being at the same time more computationally efficient. We furthermore show that several previous models for learning on dynamic graphs can be cast as specific instances of our framework. We perform a detailed ablation study of different components of our framework and devise the best configuration that achieves state-of-the-art performance on several transductive and inductive prediction tasks for dynamic graphs.

1 Introduction

In the past few years, graph representation learning [7, 27, 4] has produced a sequence of successes, gaining increasing popularity in machine learning. Graphs are ubiquitously used as models for systems of relations and interactions in many fields [5, 52, 42, 10, 16, 20, 49, 53], in particular, social sciences [68, 43] and biology [76, 62, 18]. Learning on such data is possible using graph neural networks (GNNs) [26] that typically operate by a message passing mechanism [4] aggregating information in a neighborhood of a node and create node embeddings that are then used for node-wise classification [42, 61, 35] or edge prediction [72] tasks.

The majority of methods for deep learning on graphs assume that the underlying graph is static. However, most real-life systems of interactions such as social networks or biological interactomes are *dynamic*. While it is often possible to apply static graph deep learning models [37] to dynamic graphs by ignoring the temporal evolution, this has been shown to be sub-optimal [66], and in some

Preprint. Under review.

cases, it is the dynamic structure that contains crucial insights about the system. Learning on dynamic graphs is relatively recent, and most works are limited to the setting of discrete-time dynamic graphs represented as a sequence of snapshots of the graph over time [37, 15, 69, 54, 48, 71]. Few approaches support the inductive setting of generalizing to new nodes not seen during training [46, 3, 59, 36]. Such approaches are unsuitable for interesting real world settings such as social networks, where dynamic graphs are continuous (i.e. edges can appear at any time) and evolving (i.e. new nodes join the graph continuously).

Contributions. In this paper, we first propose the generic inductive framework of Temporal Graph Networks (TGNs) operating on continuous-time dynamic graphs represented as a sequence of events, and show that many previous methods are specific instances of TGNs. Second, we propose a novel training strategy allowing the model to learn from the sequentiality of the data while maintaining highly efficient parallel processing. We show that this leads to an order of magnitude speed up over previous methods. Third, we perform a detailed ablation study of different components of our framework and analyze the tradeoff between speed and accuracy. Finally, we show state-of-the-art performance on multiple tasks and datasets in both transductive and inductive settings, while being much faster than previous methods.

2 Background

Deep learning on static graphs. A static graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ comprises nodes $\mathcal{V} = \{1, \ldots, n\}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, which are endowed with features, denoted by $\mathbf{v}_i$ and $\mathbf{e}_{ij}$ for all $i, j = 1, \ldots, n$, respectively. A typical *graph neural network* (GNN) creates an *embedding* $\mathbf{z}_i$ of the nodes by learning a local aggregation rule of the form

$$\mathbf{z}_i = \sum_{j \in \mathcal{N}_i} h(\mathbf{m}_{ij}, \mathbf{v}_i) \qquad \mathbf{m}_{ij} = \mathrm{msg}(\mathbf{v}_i, \mathbf{v}_j, \mathbf{e}_{ij}),$$

which is interpreted as message passing from the neighbors j of i. Here, $\mathcal{N}_i = \{j : (i, j) \in \mathcal{E}\}$ denotes the neighborhood of node i and msg and h are learnable functions.

Dynamic Graphs. There exist two main classes of dynamic graphs. *Discrete-time dynamic graph*s (DTDG) are sequences of static graph snapshots taken at intervals in time. *Continuos-time dynamic graph*s (CTDG) are more general and can be represented as timed lists of events, which may include edge addition or deletion, node addition or deletion and node or edge feature transformations. In this paper, we do not consider deletion events.

Our temporal (multi-)graph is modeled as a sequence of time-stamped *events* $\mathcal{G} = \{x(t_1), x(t_2), \ldots\}$, representing addition or change of a node or interaction between a pair of nodes at times $0 \leq t_1 \leq t_2 \leq \ldots$. An event $x(t)$ can be of two types: 1) A **node-wise event** is represented by $\mathbf{v}_i(t)$, where i denotes the index of the node and $\mathbf{v}$ is the vector attribute associated with the event. After its first appearance, a node is assumed to live forever and its index is used consistently for the following events. 2) An **interaction event** between nodes i and j is represented by a (directed) *temporal edge* $\mathbf{e}_{ij}(t)$ (there might be more than one edge between a pair of nodes, so technically this is a multigraph). We denote by $\mathcal{V}(T) = \{i : \exists \mathbf{v}_i(t) \in \mathcal{G}, t \in T\}$ and $\mathcal{E}(T) = \{(i, j) : \exists \mathbf{e}_{ij}(t) \in \mathcal{G}, t \in T\}$ the temporal set of vertices and edges, respectively, and by $\mathcal{N}_i(T) = \{j : (i, j) \in \mathcal{E}(T)\}$ the neighborhood of node i in time interval T. $\mathcal{N}_i^k(T)$ denotes the k-hop neighborhood. A *snapshot* of the temporal graph $\mathcal{G}$ at time t is the (multi-)graph $\mathcal{G}(t) = (\mathcal{V}[0, t], \mathcal{E}[0, t])$ with $n(t)$ nodes.

3 Temporal Graph Networks

Following the terminology in [32], a neural model for dynamic graphs can be regarded as an encoder-decoder pair, where an encoder is a function that maps from a dynamic graph to node embeddings, and a decoder takes as input one or more node embeddings and makes a prediction based on these, e.g. node classification or edge prediction. The key contribution of this paper is a novel Temporal Graph Network (TGN) encoder applied on a continuous-time dynamic graph represented as a sequence of time-stamped events and producing, for each time t, the embedding of the graph nodes $\mathbf{Z}\left(t\right) = \left(\mathbf{z}_1(t), \ldots, \mathbf{z}_{n(t)}(t)\right)$.

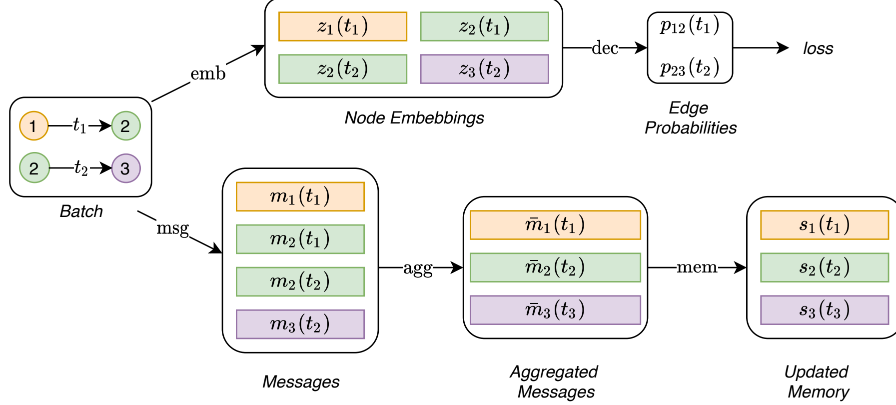


Figure 1: Two flows of operations for processing a batch of time-stamped interactions using TGN. *Top:* using the embedding module to compute the temporal node embeddings and subsequently the loss function. *Bottom:* memory update from batch interactions.

3.1 Core modules

Memory. The memory (state) of the model at time t consists of a vector $\mathbf{s}_i(t)$ for each node i the model has seen so far. The memory of a node is updated when the node is involved in an event (e.g. interaction with an other node or node-wise change), and its purpose is to represent the history of a node in a compressed format. Thanks to this specific module, TGNs have the capability to memorize long term dependencies for each node in the graph.

In addition, a global memory can be added to the model to track the evolution of the entire temporal network. While we envisage the benefits such a memory could bring (e.g. information can easily travel long distances in the graph, nodes' memory can be updated w.r.t the changes in global state, easy graph-wise predictions based on global memory), such a direction has not been explored in this work and it is as such left to future research.

Message Function. For each event involving node i, a message is computed to update i's memory. In the case of an interaction event $\mathbf{e}_{ij}(t)$ between nodes i and j at time t, two messages can be computed for the source and target nodes that respectively start and receive the interaction:

$$\mathbf{m}_i(t) = \mathrm{msg_s}\left(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), t, \mathbf{e}_{ij}(t)\right), \qquad \mathbf{m}_j(t) = \mathrm{msg_d}\left(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), t, \mathbf{e}_{ij}(t)\right) \tag{1}$$

Similarly, in case of a node-wise event $\mathbf{v}_i(t)$, a single message can be computed for the node involved in the event:

$$\mathbf{m}_i(t) = \mathrm{msg_n}\left(\mathbf{s}_i(t^-), t, \mathbf{v}_i(t)\right). \tag{2}$$

Here, $\mathbf{s}_i(t^-)$ is the memory of node i just before time t, and $\mathrm{msg_s}, \mathrm{msg_d}$ and $\mathrm{msg_n}$ are learnable message functions, e.g. MLPs. In all our experiments, we chose the message function as *identity* (id), which is simply the concatenation of the inputs, for the sake of simplicity.

Message Aggregator. Resorting to batch processing for efficiency reasons may lead to multiple events involving the same node i in the same batch. As each event generates a message in our formulation, we use a mechanism to aggregate messages $\mathbf{m}_i(t_1), \ldots, \mathbf{m}_i(t_b)$ for $t_1, \ldots, t_b \leq t$,

$$\bar{\mathbf{m}}_i(t) = \mathrm{agg}\left(\mathbf{m}_i(t_1), \ldots, \mathbf{m}_i(t_b)\right). \tag{3}$$

Here, agg is an aggregation function. While multiple choices can be considered for implementing this module (e.g. RNNs or attention w.r.t. the node memory), for the sake of simplicity we considered two efficient non-learnable solutions in our experiments: *most recent message* (keep only most recent message for a given node) and *mean message* (average all messages for a given node). We leave learnable aggregation as a future research direction.

Memory Updater. As previously mentioned, the memory of a node is updated upon each event involving the node itself:

$$\mathbf{s}_i(t) = \mathrm{mem}\left(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)\right). \tag{4}$$

For interaction events involving two nodes i and j, the memories of both nodes are updated after the event has happened. For node-wise events, only the memory of the related node is updated. Here, mem is a learnable memory update function, e.g. a recurrent neural network such as LSTM [29] or GRU [9].

Embedding. The embedding module is used to generate the temporal embedding $\mathbf{z}_i(t)$ of node i at any time t. The main goal of the embedding module is to avoid the so-called memory staleness problem [32]. Since the memory of a node i is updated only when the node is involved in an event, it might happen that, in the absence of events for a long time (e.g. a social network user who stops using the platform for some time before becoming active again), i's memory becomes stale. While multiple implementations of the embedding module are possible, we use the form:

$$\mathbf{z}_i(t) = \mathrm{emb}(i,t) = \sum_{j \in n_i^k([0,t])} h\left(\mathbf{s}_i(t), \mathbf{s}_j(t), \mathbf{e}_{ij}, \mathbf{v}_i(t), \mathbf{v}_j(t)\right),$$

where h is a learnable function. This includes many different formulations as particular cases:

Identity (id): $\mathrm{emb}(i,t) = \mathbf{s}_i(t)$, which uses the memory directly as the node embedding.

Time projection (time): $\mathrm{emb}(i,t) = (1 + \Delta t\, \mathbf{w}) \circ \mathbf{s}_i(t)$, where $\mathbf{w}$ are learnable parameters, Δt is the time since the last interaction, and $\circ$ denotes element-wise vector product. This version of the embedding method was used in JODIE [36].

Temporal Graph Attention (attn): A series of L graph attention layers compute i's embedding by aggregating information from its L-hop temporal neighborhood. The input to the l-th layer is i's representation $\mathbf{h}_i^{(l-1)}(t)$, the current timestamp t, i's neighborhood representation $\{\mathbf{h}_1^{(l-1)}(t), \ldots, \mathbf{h}_N^{(l-1)}(t)\}$ together with timestamps $t_1, \ldots, t_N$ and features $\mathbf{e}_{i1}(t_1), \ldots, \mathbf{e}_{iN}(t_N)$ for each of the considered interactions which form an edge in i's temporal neighborhood:

$$\mathbf{h}_i^{(l)}(t) = \mathrm{MLP}^{(l)}(\mathbf{h}_i^{(l-1)}(t) \,\|\, \tilde{\mathbf{h}}_i^{(l)}(t)), \tag{5}$$

$$\tilde{\mathbf{h}}_i^{(l)}(t) = \mathrm{MultiHeadAttention}^{(l)}(\mathbf{q}^{(l)}(t), \mathbf{K}^{(l)}(t), \mathbf{V}^{(l)}(t)), \tag{6}$$

$$\mathbf{q}^{(l)}(t) = \mathbf{h}_i^{(l-1)}(t) \,\|\, \boldsymbol{\phi}(0), \tag{7}$$

$$\mathbf{K}^{(l)}(t) = \mathbf{V}^{(l)}(t) = \mathbf{C}^{(l)}(t), \tag{8}$$

$$\mathbf{C}^{(l)}(t) = [\mathbf{h}_1^{(l-1)}(t) \,\|\, \mathbf{e}_{i1}(t_1) \,\|\, \boldsymbol{\phi}(t - t_1), \ldots, \mathbf{h}_N^{(l-1)}(t) \,\|\, \mathbf{e}_{iN}(t_N) \,\|\, \boldsymbol{\phi}(t - t_N)]. \tag{9}$$

Here, $\boldsymbol{\phi}(\cdot)$ represents a generic time encoding [66], $\|$ is the concatenation operator and $\mathbf{z}_i(t) = \mathrm{emb}(i,t) = \mathbf{h}_i^{(L)}(t)$. Each layer amounts to performing multi-head-attention [60] where the query ($\mathbf{q}^{(l)}(t)$) is a reference node (i.e. the target node or one of its $L-1$-hop neighbors), and the keys $\mathbf{K}^{(l)}(t)$ and values $\mathbf{V}^{(l)}(t)$ are its neighbors. Finally, an MLP is used to combine the reference node representation with the aggregated information. Differently from the original formulation of this layer (firstly proposed in TGAT [66]) where no node-wise temporal features were used, in our case the input representation of each node $\mathbf{h}_j^{(0)}(t) = \mathbf{s}_j(t) + \mathbf{v}_j(t)$ and as such it allows the model to exploit both the current memory $\mathbf{s}_j(t)$ and the temporal node features $\mathbf{v}_j(t)$.

Temporal Graph Sum (sum): A simpler and faster aggregation over the graph:

$$\mathbf{h}_i^{(l)}(t) = \mathrm{MLP}^{(l)}(\mathbf{h}_i^{(l-1)}(t) \,\|\, \tilde{\mathbf{h}}_i^{(l)}(t)), \tag{10}$$

$$\tilde{\mathbf{h}}_i^{(l)}(t) = \sum_{j \in n_i([0,t])} \mathbf{h}_j^{(l-1)}(t) \,\|\, \mathbf{e}_{ij} \,\|\, \boldsymbol{\phi}(t - t_j). \tag{11}$$

Here as well, $\boldsymbol{\phi}(\cdot)$ is a time encoding and $\mathbf{z}_i(t) = \mathrm{emb}(i,t) = \mathbf{h}_i^{(L)}(t)$.

3.2 Training

Our TGN model can be trained for a variety of tasks such as future edge prediction (self-supervised setting) or node classification (semi-supervised setting). We present two possible training procedures

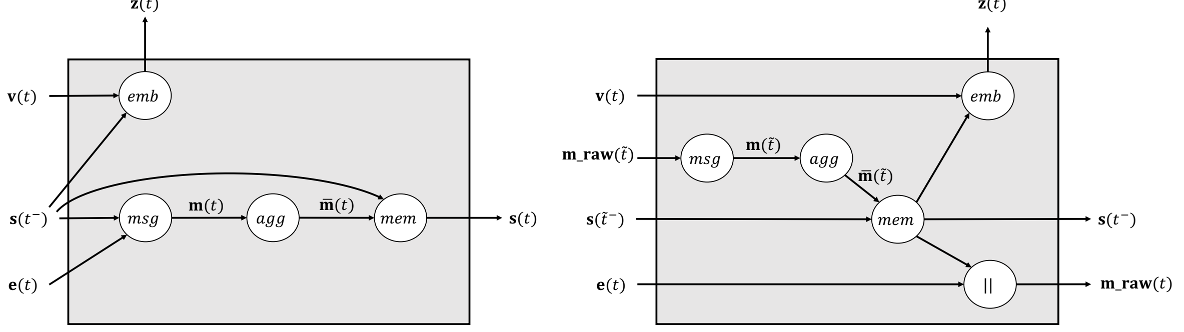


Figure 2: Two implementations of TGN with different memory updates. *Left:* Basic training strategy. *Right:* Advanced training strategy. $\mathbf{m_raw}(t)$ is the raw message generated by event $\mathbf{e}(t)$, $\tilde{t}$ is the instant of time of the last event involving each node, and t^- the one immediately preceding t.

for TGNs while using the link prediction task as a simple example: provided a list of ordered timed interactions, the goal of the model is to predict the future interactions from those observed in the past. Both training procedures are detailed in Algorithms 1 and 2, and Figure 2 depicts how TGN modules are combined.

Figure 1 shows that interactions serve two purposes: 1) they are the training objective, 2) they are used to update the memory. While the interactions in a batch cannot be used to update the memory before predicting the same interactions (as this would leak information), reversing the order of the operations, i.e. predicting the interactions and computing the loss before updating the memory, causes all memory-related modules (**Message Function**, **Message Aggregator**, and **Memory Updater**) not to receive a gradient (Algorithm 1). Therefore, extra steps must be taken in order to train these modules.

Algorithm 1: Training TGN - No gradient flows

1. $\mathbf{s} \leftarrow \mathbf{0}$; `// Initialize memory to zeros`
2. **foreach** *batch* $(\mathbf{i}, \mathbf{j}, \mathbf{e}, \mathbf{t}) \in$ *training data* **do**
3. $\quad \mathbf{n} \leftarrow$ sample negatives ;
4. $\quad \mathbf{z_i}, \mathbf{z_j}, \mathbf{z_n} \leftarrow$ emb$(\mathbf{i}, \mathbf{t})$, emb$(\mathbf{j}, \mathbf{t})$, emb$(\mathbf{n}, \mathbf{t})$; `// Compute node embeddings`
5. $\quad \mathbf{p_{pos}}, \mathbf{p_{neg}} \leftarrow \mathrm{dec}(\mathbf{z_i}, \mathbf{z_j}), \mathrm{dec}(\mathbf{z_i}, \mathbf{z_n})$; `// Compute interactions probs`
6. $\quad l$ = BCE$(\mathbf{p_{pos}}, \mathbf{p_{neg}})$; `// Compute BCE loss`
7. $\quad \mathbf{m_i}, \mathbf{m_j} \leftarrow$ msg$(\mathbf{s_i}, \mathbf{s_j}, \mathbf{t}, \mathbf{e})$, msg$(\mathbf{s_j}, \mathbf{s_i}, \mathbf{t}, \mathbf{e})$; `// Compute messages`[1]
8. $\quad \bar{\mathbf{m}} \leftarrow$ agg$(\mathbf{m_i} || \mathbf{m_j})$; `// Aggregate messages for the same nodes`
9. $\quad \mathbf{s_i}, \mathbf{s_j} \leftarrow$ mem$(\bar{\mathbf{m}}_\mathbf{i}, \mathbf{s_i})$, mem$(\bar{\mathbf{m}}_\mathbf{j}, \mathbf{s_j})$; `// Update memory`
10. **end**

Basic training strategy. The simplest strategy keeps the same order of operations as Algorithm 1 (predict interactions, then update memory), but breaks every batch[2] of size b into k sub-batches of size b/k. The sub-batches are processed sequentially with their losses accumulated and backpropagation is only performed after the last sub-batch. If a node appears in two sub-batches, its memory in the second sub-batch will depend on the computation done by the memory-related modules in the first. Therefore, these modules will receive a gradient.

Advanced training strategy. While the basic training procedure is straightforward to implement, it presents two drawbacks: 1) it slows down the training, as each batch is not computed fully in parallel, 2) the only nodes that contribute to the memory-related modules' gradients are those with at least one interaction in multiple sub-batches. Therefore, these modules can still receive no gradient if sub-batches do not share any nodes, or the gradient can be heavily skewed towards a few nodes that

[2]By 'batch' we refer to what is sometime defined as mini-batch, i.e. a subset of the original dataset, which is used for mini-batch gradient descent.

appear multiple times, leading to biased update steps and ultimately to a sub-optimal local minimum for the overall training procedure.

The solution to this problem is to reverse the order of operations. Let $\tilde{t}_i$ be the time of node i's last interaction in its last sub-batch $b_i(\tilde{t}_i)$. Instead of letting the memory be representative of the entire set of interactions involving i in the past, we store memory $\mathbf{s}_i(\tilde{t}_i^-)$, i.e. the state of i prior to the last sub-batch $b_i(\tilde{t}_i)$, together with the raw information we need to update $\mathbf{s}_i(\tilde{t}_i^-)$ with the interactions of $b_i(\tilde{t}_i)$ (i.e. the set of raw update messages $\{(\mathbf{s}_i(\tilde{t}_i^-), \mathbf{s}_j(\tilde{t}_i^-), \mathbf{e}_{ij}(t), t) \;\; \forall \mathbf{e}_{ij}(t) \in b_i(\tilde{t}_i)\}$ of i's interactions in $b_i(\tilde{t}_i)$). At the beginning of each sub-batch, the model first updates the nodes' memories by computing and aggregating messages from the stored raw information (line 11 of Algorithm 2), then uses the updated memory to infer the embeddings and computes the loss function (Figure 2 right). As a result, the loss function depends on a memory which has just been updated by its related modules. Moreover, all nodes involved in the computation of the embeddings (i.e. all source and target nodes and related neighbors) contribute to the gradients, ultimately producing more stable optimization and better local minima (Figure 3b).

While the advanced training strategy is sufficient to train TGNs, it can also be combined with the basic strategy by breaking each batch into sub-batches. We investigate the speed vs accuracy tradeoff of different combinations of the two strategies in Section 5.

Algorithm 2: Training TGN - Advanced Strategy

```
1  s ← 0 ;                                                      // Initialize memory to zeros
2  m_raw ← {} ;                                                 // Initialize raw messages
3  foreach batch (i, j, e, t) ∈ training data do
4      n ← sample negatives ;
5      m ← msg(m_raw) ;                                         // Compute messages from raw features¹
6      m̄ ← agg(m) ;                                            // Aggregate messages for the same nodes
7      ŝ ← mem(m̄, s) ;                                         // Get updated memory
8      z_i, z_j, z_n ← emb_ŝ(i, t), emb_ŝ(j, t), emb_ŝ(n, t) ;  // Compute node embeddings³
9      p_pos, p_neg ← dec(z_i, z_j), dec(z_i, z_n) ;            // Compute interactions probs
10     l = BCE(p_pos, p_neg) ;                                  // Compute BCE loss
11     m_raw_i, m_raw_j ← (ŝ_i, ŝ_j, t, e), (ŝ_j, ŝ_i, t, e) ;  // Compute raw messages
12     s_i, s_j ← ŝ_i, ŝ_j ;                                    // Store updated memory for sources and destinations
13 end
```

4 Related Work

Early models for learning on dynamic graphs focused on Discrete Time Dynamic Graphs (DTDG)s. Such approaches either aggregate graph snapshots and then apply static methods [37, 28, 56, 31, 1, 2], assemble snapshots into tensors and factorize [15, 70, 39], or encode each snapshot to produce a series of embeddings. In the latter case, the embeddings are either aggregated by taking a weighted sum [67, 74], fit to time series models [30, 24, 11, 44], used as components in RNNs [55, 45, 41, 69, 8, 54, 48], or learned by imposing a smoothness constraint over time [33, 25, 67, 75, 73, 57, 22, 17, 50]. Another line of work encodes DTDGs by first performing random walks on an initial snapshot and then modifying the walk behaviour for subsequent snapshots [40, 14, 64, 13, 71].

Only recently have Continuous Time Dynamic Graphs (CTDGs) been addressed. Several approaches use random walk models [47, 46, 3] that incorporate continuous time through constraints on transition probabilities. Sequence-based approaches for CTDGs [36, 58, 59, 38] use RNNs to update representations of the source and destination node each time a new edge appears. Other recent works have focused on dynamic knowledge graphs [21, 65, 12, 19].

Most recent CTDG learning models can be interpreted as specific cases of our framework (see Table 1). For example, Jodie [36] uses the time projection embedding module $\mathrm{emb}(i,t) = (1+\Delta t\mathbf{w}) \circ \mathbf{s}_i(t)$. TGAT [66] is a specific case of TGN when the memory and its related modules are missing, and

[1] For the sake of clarity, we use the same message function for both sources and destination.

[3] We denote with $\mathrm{emb}_{\hat{\mathbf{s}}}$ an embedding layer that operates on the updated version of the memory $\hat{\mathbf{s}}$.

Table 1: Previous models for deep learning on continuous-time dynamic graphs are specific case of our TGN framework. Shown are multiple variants of TGN used in our ablation studies. *method* (l,n) refers to graph convolution using l layers and n neighbors. †uses t-batches. * uses uniform sampling of neighbors, while the default is sampling the most recent neighbors.

	Mem.	Mem. Update	Embedding	Mess. Agg.	Mess. Func.
JODIE	node	RNN	time	—†	id
TGAT	—	—	attn (2l, 20n)*	—	—
TGN-attn	node	GRU	attn (1l, 10n)	last	id
TGN-2l	node	GRU	attn (2l, 10n)	last	id
TGN-no-mem	—	—	attn (1l, 10n)	—	id
TGN-time	node	GRU	time	last	id
TGN-id	node	GRU	id	last	id
TGN-sum	node	GRU	sum (1l, 10n)	last	id
TGN-mean	node	GRU	attn (1l, 10n)	mean	id

graph attention is used as the Embedding module. Finally, we note that TGN generalizes the Graph Networks (GN) model [4] for static graphs (with the exception of the global block that we mentioned before), and thus the majority of existing message passing-type architectures.

For additional background, we refer the reader to surveys on general graph representation learning [7, 27, 4] and the recent survey on dynamic graph learning [32].

5 Experiments

5.1 Experimental Settings

Datasets. We use three datasets in our experiments: Wikipedia, Reddit [36], and Twitter. Reddit and Wikipedia are bipartite interaction graphs. In the Reddit dataset, users and sub-reddits are nodes, and an interaction occurs when a user writes a post to the sub-reddit. In the Wikipedia dataset, users and pages are nodes, and an interaction represents a user editing a page. In both aforementioned datasets, the interactions are represented by text features (of a post or page edit, respectively), and labels represent whether a user is banned. Both interactions and labels are time-stamped.

The Twitter dataset is a non-bipartite graph released as part of the 2020 RecSys Challenge [6]. Nodes are users and interactions are retweets. The features of an interaction are a BERT-based [63] vector representation of the text of the retweet. We use a subset of the original dataset formed by taking the largest connected component of the retweet graph and retaining only the nodes with the 5000 highest in degrees and 5000 highest out degrees. Dataset statistics together with more details are provided in the supplementary material.

Tasks. Our experimental setup closely follows [66] and focuses on the tasks of future edge prediction and dynamic node classification. On the former, we use both the transductive and inductive settings. In the transductive task, we predict future links of nodes which were observed during training, whereas in the inductive tasks we predict future links of nodes never observed before. The transductive setting is used for node classification. We perform the same 70%-15%-15% chronological split as in [66].

Future Edge Prediction. The goal is to predict the probability of an edge occurring between two nodes at a given time. Our encoder is combined with a simple MLP decoder mapping from the concatenation of two node embeddings to the probability of the edge. For the Wikipedia and Reddit datasets, we use Adam optimizer with a learning rate of 0.0001, a batch size of 200 for both training, validation and testing, and early stopping with a patience of 5. For the Twitter dataset, the only change is the learning rate, which is set to 0.00005. We sample an equal amount of negatives to the positive interactions, and use *average precision* as reference metric. All results are averaged over 10 runs to obtain mean and standard deviation.

Table 2: Average Precision (%) for future edge prediction task in transductive and inductive settings. Mean and standard deviations are computed over 10 runs. *Static graph method. †Does not support inductive setting.

	Wikipedia		Reddit		Twitter	
	Transductive	Inductive	Transductive	Inductive	Transductive	Inductive
GAE*	91.44 ± 0.1	†	93.23 ± 0.3	†	—	†
VAGE*	91.34 ± 0.3	†	92.92 ± 0.2	†	—	†
DeepWalk*	90.71 ± 0.6	†	83.10 ± 0.5	†	—	†
Node2Vec*	91.48 ± 0.3	†	84.58 ± 0.5	†	—	†
GAT*	94.73 ± 0.2	91.27 ± 0.4	97.33 ± 0.2	95.37 ± 0.3	67.57 ± 0.4	62.32 ± 0.5
GraphSAGE*	93.56 ± 0.3	91.09 ± 0.3	97.65 ± 0.2	96.27 ± 0.2	65.79 ± 0.6	60.13 ± 0.6
CTDNE	92.17 ± 0.5	†	91.41 ± 0.3	†	—	†
JODIE	94.33 ± 0.4	91.29 ± 0.5	96.36 ± 0.5	94.62 ± 0.5	62.05 ± 1.0	52.72 ± 1.6
TGAT	95.34 ± 0.1	93.99 ± 0.3	98.12 ± 0.2	96.62 ± 0.3	67.84 ± 0.6	62.21 ± 0.6
TGN-attn	**98.64** ± 0.1	**98.05** ± 0.1	**98.80** ± 0.1	**97.71** ± 0.1	**93.66** ± 1.3	**90.16** ± 2.4

Table 3: ROC AUC % for the dynamic node classification. Mean and standard deviations are computed over 10 runs. *Static graph method.

	Wikipedia	Reddit
GAE*	74.85 ± 0.6	58.39 ± 0.5
VAGE*	73.67 ± 0.8	57.98 ± 0.6
GAT*	82.34 ± 0.8	64.52 ± 0.5
GraphSAGE*	82.42 ± 0.7	61.24 ± 0.6
CTDNE	75.89 ± 0.5	59.43 ± 0.6
JODIE	87.17 ± 0.5	59.50 ± 2.1
TGAT	83.69 ± 0.7	65.56 ± 0.7
TGN-attn	**88.56** ± 0.3	**68.63** ± 0.7

Table 4: Different settings of combinations of models and training strategies. #sb is number of sub-batches.

Setting	Model	Update	#sb
TGN-id-s1	TGN-id	start	1
TGN-id-s5	TGN-id	start	5
TGN-id-e1	TGN-id	end	1
TGN-id-e5	TGN-id	end	5
TGN-att-s1	TGN-att	start	1
TGN-att-s5	TGN-att	start	5
TGN-att-e1	TGN-att	end	1
TGN-att-e5	TGN-att	end	5

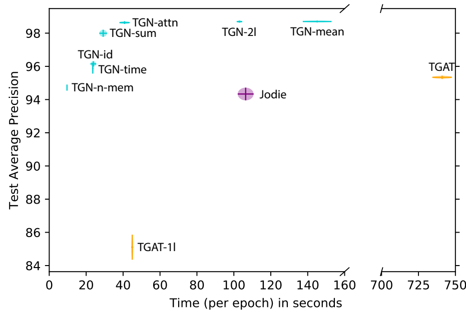
Dynamic Node Classification. The task is to predict a binary label indicating whether a user was banned at a specific time. We pre-train our encoder on the future edge prediction task, then freeze it and combine it with a task-specific MLP decoder. We use the Adam optimizer with a learning rate of 0.0003 and a batch size of 100 for both training, validation and testing. The metric used is the ROC-AUC. All results are averaged over 10 runs to obtain mean and standard deviation.

Baselines. Our strong baselines are state-of-the-art approaches for continuous time dynamic graphs (CTDNE [47], Jodie [36], and TGAT [66]) as well as state-of-the-art models for static graphs (GAE [34], VGAE [34], DeepWalk [51], Node2Vec [23], GAT [61] and GraphSAGE [27]).

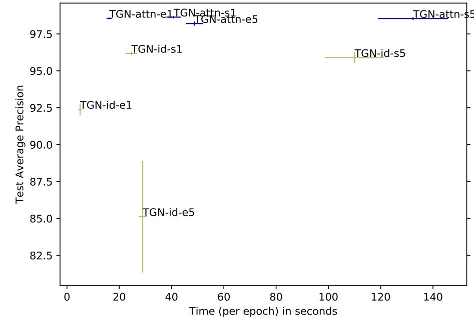
5.2 Performance

Results. Table 2 presents the results on future edge prediction. Our model clearly outperforms the baselines by a large margin in both the transductive and the inductive setting on all datasets. The gap is particularly large on the Twitter dataset, where we outperfom the second-best method (TGAT) by over 25%. Table 3 shows the results on dynamic node classification, where again our model obtains state-of-the-art results, with a large improvement over all other methods.

Speed. Due to the efficient parallel processing and the need for only one graph attention layer (see section 5.3 for the ablation study on the number of layers), our model is up to 3× faster than Jodie and about 19× faster than TGAT to complete a single epoch (see Figure 3a), while requiring a similar number of epochs to converge.



(a) Tradeoff between accuracy (test average precision in %) and speed (time per epoch in sec) of different models.



(b) Tradeoff between accuracy (test average precision in %) and speed (time per epoch in sec) of different training strategies.

Figure 3: Ablation studies on the Wikipedia dataset for the transductive setting. Means and standard deviations (visualized as ellipses) were computed over 10 runs.

5.3 Choice of Modules

We perform a detailed ablation study comparing different instances of our TGN framework. We are particular interested in the speed vs accuracy tradeoff resulting from the choice of modules and their combination. The variants we experiment with are reported in Table 1 and their results are depicted in Figure 3a.

Memory. We compare a model that does not make use of a memory (TGN-no-mem), with a model which uses memory (TGN-attn) but is otherwise identical. While TGN-att is about $3\times$ slower, it vastly outperforms TGN-no-mem, confirming the importance of memory for effective learning on dynamic graphs, due to its ability to store long-term information about a node which is otherwise hard to capture.

Embedding Module. We compared models with different embedding modules (TGN-id, TGN-time, TGN-attn, TGN-sum). The first interesting insight is that projecting the embedding in time seems to slightly hurt, as shown by the fact that TGN-time underperforms TGN-id. Moreover, the ability to exploit the graph is crucial for performance: we note that all graph-based projections (TGN-attn, TGN-sum) outperform the graph-less TGN-id model by a large margin, with TGN-attn being the top performer at the expense of being slightly slower than the simpler TGN-sum.

Message Aggregator. We compared two models, one using the most last message aggregator (TGN-attn) and another a mean aggregator (TGN-mean-aggr) but otherwise the same. While TGN-mean-aggr performs slightly better, it is more than $3\times$ slower.

Number of layers. While in TGAT having 2 layers is of fundamental importance for obtaining good performances (TGAT vs TGAT-1l), in TGN the presence of the memory makes it enough to use 1 layer to obtain very high performances (TGN-attn vs TGN-2l). This is probably because when accessing the memory of the 1-hop neighbors, we are indirectly accessing information from hops further away. Moreover, being able to use only 1 layer of graph attention speeds up the model dramatically.

5.4 Training Strategies

Table 4 shows the configurations we experimented with for this ablation study, while figure 3b presents the results. The TGN-id model makes only use of the memory (no embedding module) and therefore makes for a perfect testbed for training strategies related to the memory-related modules. Looking at the results with the TGN-id model, our proposed strategy of updating the memory at the start of the epoch clearly outperforms updating at the end.

Interestingly, when using a graph attention embedding module (TGN-attn), the benefit of the advanced strategy shrinks. This is probably due to the fact that the embedding module is able is able to adapt to the random memory-related modules, effectively denoising the spurious behavior of the nodes' memory.

6 Conclusion

We introduce TGN, a generic framework for learning on continuous-time dynamic graphs. We obtain state-of-the-art results on several tasks and datasets while being faster than previous methods. Detailed ablation studies shows the importance of the memory and its related modules to store long-term information, as well as the importance of the graph-based embedding module to generate up-to-date node embeddings. We envision interesting applications of TGN in the fields of social sciences, recommender systems, and biological interaction networks, opening up a future research direction of exploring more advanced settings of our model and understanding the most appropriate domain-specific choices.

Broader Impact

Graph Neural Networks able to effectively process temporal graphs can potentially serve a variety of purposes in our society e.g. improved recommender systems that take into account the evolving nature of users on social networks or marketplaces, as well as better filtering mechanisms for the detection of unhealthy behaviors such as spam or coordinate manipulation. At the same time, due to the novelty of these approaches, the robustness of such architectures w.r.t. external adversarial attacks has not been validated yet in the literature. Additional studies will thus need to be realised to identify the potential risks and benefits that temporal graph neural networks may present when subjected to adversarial attacks, before these can be applied to sensitive personal data and extensively exploited in industrial applications.

References

[1] N. M. Ahmed and L. Chen. An efficient algorithm for link prediction in temporal uncertain social networks. *Information Sciences*, 331:120–136, 2016.

[2] N. M. Ahmed, L. Chen, Y. Wang, B. Li, Y. Li, and W. Liu. Sampling-based algorithm for link prediction in temporal networks. *Information Sciences*, 374:1–14, 2016.

[3] N. Bastas, T. Semertzidis, A. Axenopoulos, and P. Daras. evolve2vec: Learning network representations using temporal unfolding. In *International Conference on Multimedia Modeling*, pages 447–458. Springer, 2019.

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

[5] P. W. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *NIPS*, pages 4502–4510, 2016.

[6] L. Belli, S. I. Ktena, A. Tejani, A. Lung-Yut-Fon, F. Portman, X. Zhu, Y. Xie, A. Gupta, M. M. Bronstein, A. Delić, et al. Privacy-preserving recommender systems challenge on twitter's home timeline. *arXiv:2004.13715*, 2020.

[7] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42, 2017.

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

[9] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

[10] N. Choma, F. Monti, L. Gerhardt, T. Palczewski, Z. Ronaghi, P. Prabhat, W. Bhimji, M. M. Bronstein, S. Klein, and J. Bruna. Graph neural networks for icecube signal classification. In *ICMLA*, 2018.

[11] P. R. da Silva Soares and R. B. C. Prudêncio. Time series based link prediction. In *IJCNN*, pages 1–7. IEEE, 2012.

[12] S. S. Dasgupta, S. N. Ray, and P. Talukdar. HyTE: Hyperplane-based temporally aware knowledge graph embedding. In *EMNLP*, pages 2001–2011, 2018.

[13] S. De Winter, T. Decuypere, S. Mitrović, B. Baesens, and J. De Weerdt. Combining temporal aspects of dynamic networks with node2vec for a more efficient dynamic link prediction. In *ASONAM*, pages 1234–1241, 2018.

[14] L. Du, Y. Wang, G. Song, Z. Lu, and J. Wang. Dynamic network embedding: An extended approach for skip-gram based network embedding. In *IJCAI*, pages 2086–2092, 2018.

[15] D. M. Dunlavy, T. G. Kolda, and E. Acar. Temporal link prediction using matrix and tensor factorizations. *TKDD*, 5(2):1–27, 2011.

[16] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*. 2015.

[17] A. M. Fard, E. Bagheri, and K. Wang. Relationship prediction in dynamic heterogeneous information networks. In *European Conference on Information Retrieval*, pages 19–34. Springer, 2019.

[18] P. Gainza et al. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods*, 17:184–192, 2019.

[19] A. García-Durán, S. Dumančić, and M. Niepert. Learning sequence encoders for temporal knowledge graph completion. In *EMNLP*, pages 4816–4821, 2018.

[20] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[21] R. Goel, S. M. Kazemi, M. Brubaker, and P. Poupart. Diachronic embedding for temporal knowledge graph completion. *arXiv:1907.03143*, 2019.

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

[23] A. Grover and J. Leskovec. Node2vec: Scalable feature learning for networks. In *KDD '16*, KDD '16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.

[24] İ. Güneş, Ş. Gündüz-Öğüdücü, and Z. Çataltepe. Link prediction using time series of neighborhood-based node similarity scores. *Data Mining and Knowledge Discovery*, 30(1):147–180, 2016.

[25] M. Gupta, C. C. Aggarwal, J. Han, and Y. Sun. Evolutionary clustering and analysis of bibliographic networks. In *ASONAM*, pages 63–70. IEEE, 2011.

[26] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[27] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *arXiv:1709.05584*, 2017.

[28] R. Hisano. Semi-supervised graph embedding approach to dynamic link prediction. *Springer Proceedings in Complexity*, page 109–121, 2018.

[29] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[30] Z. Huang and D. K. Lin. The time-series link prediction problem with applications in communication surveillance. *INFORMS Journal on Computing*, 21(2):286–303, 2009.

[31] N. M. A. Ibrahim and L. Chen. Link prediction in dynamic social networks by integrating different types of information. *Applied Intelligence*, 42(4):738–750, 2015.

[32] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

[33] M.-S. Kim and J. Han. A particle-and-density based evolutionary clustering method for dynamic networks. *VLDB*, 2(1):622–633, 2009.

[34] T. N. Kipf and M. Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

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

[36] S. Kumar, X. Zhang, and J. Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD '19*, page 1269–1278, 2019.

[37] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 58(7):1019–1031, May 2007.

[38] Y. Ma, Z. Guo, Z. Ren, E. Zhao, J. Tang, and D. Yin. Streaming graph neural networks. *arXiv:1810.10627*, 2018.

[39] Y. Ma, V. Tresp, and E. A. Daxberger. Embedding models for episodic knowledge graphs. *Journal of Web Semantics*, 59:100490, 2019.

[40] S. Mahdavi, S. Khoshraftar, and A. An. dynnode2vec: Scalable dynamic network embedding. In *2018 IEEE International Conference on Big Data*, pages 3762–3765. IEEE, 2018.

[41] F. Manessi, A. Rozza, and M. Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000, 2020.

[42] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *CVPR*, 2016.

[43] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein. Fake news detection on social media using geometric deep learning. *arXiv:1902.06673*, 2019.

[44] B. Moradabadi and M. R. Meybodi. A novel time series link prediction method: Learning automata approach. *Physica A: Statistical Mechanics and its Applications*, 482:422–432, 2017.

[45] A. Narayan and P. H. Roe. Learning graph dynamics using deep neural networks. *IFAC-PapersOnLine*, 51(2):433–438, 2018.

[46] G. H. Nguyen, J. Boaz Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim. Dynamic network embeddings: From random walks to temporal random walks. In *2018 IEEE International Conference on Big Data*, pages 1085–1092, 2018.

[47] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim. Continuous-time dynamic network embeddings. In *WWW '18*, page 969–976, 2018.

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

[49] S. Parisot, S. I. Ktena, E. Ferrante, M. Lee, R. Guerrero, B. Glocker, and D. Rueckert. Disease prediction using graph convolutional networks: Application to autism spectrum disorder and alzheimer's disease. *Med Image Anal*, 48:117–130, 2018.

[50] Y. Pei, J. Zhang, G. Fletcher, and M. Pechenizkiy. Node classification in dynamic social networks. *AALTD*, page 54, 2016.

[51] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *KDD '14*, page 701–710, 2014.

[52] S. Qi, W. Wang, B. Jia, J. Shen, and S.-C. Zhu. Learning human-object interactions by graph parsing neural networks. In *ECCV*, pages 401–417, 2018.

[53] E. Rossi, F. Monti, M. M. Bronstein, and P. Liò. ncrna classification with graph convolutional networks. In *KDD Workshop on Deep Learning on Graphs*, 2019.

[54] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *WSDM*, pages 519–527, 2020.

[55] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson. Structured sequence modeling with graph convolutional recurrent networks. *Lecture Notes in Computer Science*, page 362–373, 2018.

[56] U. Sharan and J. Neville. Temporal-relational classifiers for prediction in evolving domains. In *ICDM*, pages 540–549. IEEE, 2008.

[57] U. Singer, I. Guy, and K. Radinsky. Node embedding over temporal graphs. In *IJCAI*, pages 4605–4612, 7 2019.

[58] R. Trivedi, H. Dai, Y. Wang, and L. Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *ICML*, page 3462–3471, 2017.

[59] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha. Dyrep: Learning representations over dynamic graphs. In *ICLR*, 2019.

[60] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008. 2017.

[61] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

[62] K. Veselkov et al. Hyperfoods: Machine intelligent mapping of cancer-beating molecules in foods. *Scientific Reports*, 9(1):1–12, 2019.

[63] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew. Huggingface's transformers: State-of-the-art natural language processing. *arXiv:1910.03771*, 2019.

[64] Y. Xin, Z.-Q. Xie, and J. Yang. An adaptive random walk sampling method on dynamic community detection. *Expert Systems with Applications*, 58:10–19, 2016.

[65] C. Xu, M. Nayyeri, F. Alkhoury, J. Lehmann, and H. S. Yazdi. Temporal knowledge graph completion based on time series gaussian embedding. *arXiv:1911.07893*, 2019.

[66] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan. Inductive representation learning on temporal graphs. In *ICLR*, 2020.

[67] L. Yao, L. Wang, L. Pan, and K. Yao. Link prediction based on common-neighbors for dynamic social network. *Procedia Computer Science*, 83:82–89, 2016.

[68] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD '18*, 2018.

[69] B. Yu, M. Li, J. Zhang, and Z. Zhu. 3d graph convolutional networks with temporal graphs: A spatial information free framework for traffic forecasting. *arXiv:1903.00919*, 2019.

[70] W. Yu, W. Cheng, C. C. Aggarwal, H. Chen, and W. Wang. Link prediction with spatial and temporal consistency in dynamic networks. In *IJCAI*, pages 3343–3349, 2017.

[71] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *KDD '18*, pages 2672–2681, 2018.

[72] M. Zhang and Y. Chen. Link prediction based on graph neural networks. In *NIPS*, 2018.

[73] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang. Dynamic network embedding by modeling triadic closure process. In *AAAI*, 2018.

[74] J. Zhu, Q. Xie, and E. J. Chin. A hybrid time-series link prediction framework for large social network. In *DEXA*, pages 345–359. Springer, 2012.

[75] Y. Zhu, H. Li, Y. Liao, B. Wang, Z. Guan, H. Liu, and D. Cai. What to do next: Modeling user behaviors by time-lstm. In *IJCAI*, volume 17, pages 3602–3608, 2017.

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

Supplementary Material

Datasets

The statistics of the three datasets used are reported in table 5.

Hyperparameters

For all the models and datasets we used the same hyperparameters, which had been found to work well in the TGAT paper [66].

Table 5: Statistics of the datasets used in the experiments.

	Wikipedia	Reddit	Twitter
# Nodes	9,227	11,000	8,926
# Edges	157,474	672,447	130,865
# Edge features	172	172	768
# Edge features type	LIWC	LIWC	BERT
Timespan	30 days	30 days	7 days
Chronological Split	70%-15%-15%	70%-15%-15%	70%-15%-15%
# Nodes with dynamic labels	217	366	–

Table 6: Model Hyperparameters.

	Value
Memory Dimension	172
Node Embedding Dimension	100
Time Embedding Dimension	100
# Attention Heads	2
Dropout	0.1

Experimental Settings for Baselines

Our results for GAE [34], VGAE [34], DeepWalk [51], Node2Vec [23], GAT [61] and GraphSAGE [27], CTDNE [47] and TGAT [66] are taken directly from the TGAT paper [66].

For Jodie [36], we implement our own version in PyTorch, as a specific case of our framework with the temporal embedding module, and the t-batch training algorithm.