

Dynamic Graph Representation Learning



Zhiping (Patricia) Xiao

University of California, Los Angeles

Paper References

Dynamic Graph Embedding

- Motivation

- Self-Attention Mechanism

The Works in Details

- Overview

- DyRep

- DySAT

- DyGNN

- TGAT

- Conclusion

Paper References



Model Name	Paper	Code
DyRep	DyRep: Learning Representations over Dynamic Graphs (ICLR'19)	unofficial code
DySAT	DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks (WSDM'20)	on Github
DyGNN	Streaming Graph Neural Networks (SIGIR'20)	(not yet ready)
TGAT	Inductive Representation Learning on Temporal Graphs (ICLR'20)	on Github

- ▶ Based on discrete screenshot:
 - ▶ DynamicGEM (DynGEM: Deep Embedding Method for Dynamic Graphs, IJCAI'17): adopted *Net2WiderNet* and *Net2DeeperNet* approaches ($G^t = (V^t, E^t), t \in \{1, 2, \dots, T\}$) (code on Github)
 - ▶ DynamicTriad (Dynamic Network Embedding by Modeling Triadic Closure Process, AAAI'18): based on Triadic closure process etc. ($G^t = (V, E^t, W^t), t \in \{1, 2, \dots, T\}$) (code on Github)

- ▶ Based on continuous interaction:
 - ▶ HTNE (Embedding Temporal Network via Neighborhood Formation, KDD'18): using Hawkes process to model neighborhood formation (event). ($G = (V, E, A)$)
 - ▶ CTDNE (Continuous-Time Dynamic Network Embeddings, WWW'18): using Temporal Random Walk to select edges. ($G = (V, E_T, T)$) (code on Github)
 - ▶ NetWalk (NetWalk: A Flexible Deep Embedding Approach for Anomaly Detection in Dynamic Networks, KDD'18): encoding network streams. ($G(t) = (V(t), E(t))$)

Dynamic Graph Embedding



Problem: Learning dynamic node representations.

Challenges:

- ▶ Time-varying graph structures: links and node can emerge and disappear, communities are changing all the time.
 - ▶ requires the node representations capture both **structural proximity** (as in static cases) and their **temporal evolution**.
 - ▶ Time intervals of events are uneven.
- ▶ Causes of the change: can come from different aspects, e.g. in co-authorship network, research community & career stage perspectives.
 - ▶ requires modeling multi-faceted variations.

Static graphs are often defined as:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$

However, there isn't an unified way of defining the dynamic graphs.

Reference: Attention Is All You Need (NeurIPS'17)

An **attention mechanism** has a set of keys and a set of values; it receives a sets of queries.

- ▶ (key, value) are paired up, one key matched to one value.
- ▶ queries compared to keys, seek for the corresponding values. (e.g. dictionary)

Does not require an exact match; estimate the strength of the query-key match (e.g., cosine similarity)

Assumption: more similar keys provide more reliable values.

Idea:

1. Compute the similarities between each query and **all** of the keys.
2. Compute a **weighted** average of the corresponding values, as the result.

Normally we use dot-product attention, say,

$$\text{Att}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V.$$

When the key, the value, the query are exactly the same, $\mathbf{K} = \mathbf{Q} = \mathbf{V}$, the attention is “**self-attention**”.

In practice we want more **flexible** self-attention.

We want different dimensions of a vector to have different importance when calculating the attention.

For example, we can apply the linear transformation

$$\mathbf{K} = \mathbf{XW}_\mathbf{K}, \quad \mathbf{Q} = \mathbf{XW}_\mathbf{Q}, \quad \mathbf{V} = \mathbf{XW}_\mathbf{V},$$

to the key, query and value of a self-attention mechanism.

Free parameters $\mathbf{W}_\mathbf{K}, \mathbf{W}_\mathbf{Q}, \mathbf{W}_\mathbf{V}$ bring a lot of randomness to the attention mechanism. Normally, we want to simultaneously try multiple sets of weights. That makes a “**multi-head self-attention**”.

Given h sets of weights (which constitute h heads), we write

$$\begin{aligned}\mathbf{K}_1 &= \mathbf{XW}_{\mathbf{K}_1}, & \mathbf{Q}_1 &= \mathbf{XW}_{\mathbf{Q}_1}, & \mathbf{V}_1 &= \mathbf{XW}_{\mathbf{V}_1}, \\ \mathbf{K}_2 &= \mathbf{XW}_{\mathbf{K}_2}, & \mathbf{Q}_2 &= \mathbf{XW}_{\mathbf{Q}_2}, & \mathbf{V}_2 &= \mathbf{XW}_{\mathbf{V}_2}, \\ & \vdots & & & \\ \mathbf{K}_h &= \mathbf{XW}_{\mathbf{K}_h}, & \mathbf{Q}_h &= \mathbf{XW}_{\mathbf{Q}_h}, & \mathbf{V}_h &= \mathbf{XW}_{\mathbf{V}_h},\end{aligned}$$

and we obtain h sets of results. h is called the “**head number**”, and each set of result comes from a **head**.

Using the h results together, we can improve the reliability of the self-attention mechanism (e.g., by taking an average).

The Works in Details



Model	time steps	directed	relation types
DyRep	Continuous	No	Homogeneous (*)
DySAT	Discrete	No	Homogeneous
DyGNN	Discrete	Yes	Homogeneous
TGAT	Continuous	Yes	Homogeneous

Two kinds of events: (1) association (2) communication.

$$\mathcal{G}^t = (\mathcal{V}, \mathcal{E}^t)$$

is the denotation of **undirected** graph \mathcal{G} at time $t \in [t_0, T]$. An event (u, v, t, k) has u, v being the involved nodes, $t \in \mathbb{R}^+$ be the time and $k = \{0, 1\}$ be the event type (0 for association, 1 for communication). The stream of event-observations are (evolution of graph):

$$\mathcal{O} = \{(u, v, t, k)_p\}_{p=1}^P$$

Embedding of node v at time t is denoted as $\mathbf{z}^v(t) \in \mathbb{R}^d$. $\mathbf{z}^v(\bar{t})$ represents the *most recently updated embedding* of node v just before t .

Idea: learn functions to compute node embeddings.

Examples of the two kinds of events:

- ▶ association: being academic friends
- ▶ communication: meeting at a conference

Given an event $p = (u, v, t, k)$, the conditional intensity function $\lambda_k^{u,v}$ is defined as:

$$\lambda_k^{u,v}(t) = f_k(g_k^{u,v}(\bar{t}))$$

where \bar{t} signifies the time point just before current event, and

$$g_k^{u,v}(\bar{t}) = \boldsymbol{\omega}_k^T [\mathbf{z}^u(\bar{t}); \mathbf{z}^v(\bar{t})]$$

is a function of node representations learned through GNN.

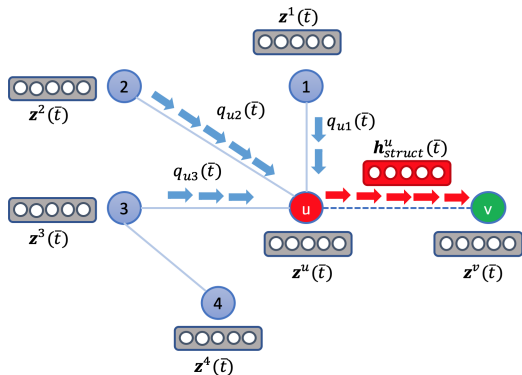
$$f_k(x) = \psi_k \log(1 + \exp(x/\psi_k))$$

$\psi_k > 0$ is scalar time-scale parameter to learn, corresponding to the rate of events. $\boldsymbol{\omega}_k$ is also a parameter to learn.

$$\mathbf{z}^v(t_p) = \sigma\left(\underbrace{\mathbf{W}^{struct} \mathbf{h}_{struct}^u(\bar{t}_p)}_{\text{Localized Embedding Propagation}} + \underbrace{\mathbf{W}^{rec} \mathbf{z}^v(\bar{t}_p^v)}_{\text{Self-Propagation}} + \underbrace{\mathbf{W}^t(t_p - \bar{t}_p^v)}_{\text{Exogenous Drive}} \right)$$

where $\mathbf{h}_{struct}^u(\bar{t}_p) \in \mathbb{R}^d$ is obtained from aggregating node u 's neighbors, $\mathbf{W}^{struct}, \mathbf{W}^{rec} \in \mathbb{R}^{d \times d}$, $\mathbf{W}^t \in \mathbb{R}^d$.

It also inherits the GAT-style multi-head attention.



Temporal Point Process Self-Attention:

$$\mathbf{h}_{struct}^u(\bar{t}) = \max(\{\sigma(q_{ui}(\bar{t}) * \mathbf{h}^i(\bar{t}))\})$$

$$\mathbf{h}^i(\bar{t}) = \mathbf{W}^h \mathbf{z}^i(\bar{t}) + b^h$$

where $i \in N_u(\bar{t})$ is the node in neighborhood of node u .

$$q_{ui}(\bar{t}) = \frac{\exp(S_{ui}(\bar{t}))}{\sum_{i' \in N_u(\bar{t})} \exp(S_{ui'}(\bar{t}))}$$

Figure: DyRep computes the temporally evolving attention based on events. q is an attention coefficient function, parameterized by \mathcal{S} , which is computed using the intensity of events between connected nodes.

This model obtains two set of parameters to be updated:

- ▶ $\mathbf{A}(t) \in \mathbb{R}^{n \times n}$, the adjacency matrix of \mathcal{G}_t . $\mathbf{A}_{uv}(t) \in \{0, 1\}$. Updated only in association events.
- ▶ $\mathcal{S}(t) \in \mathbb{R}^{n \times n}$, the stochastic matrix, denoting the likelihood of communication between each pair of nodes. $\mathcal{S}_{uv}(t) \in [0, 1]$. Updated according to $\lambda_k^{u,v}(t)$.

$$L = - \sum_{p=1}^P \log \left(\lambda_p(t) + \int_0^T \Lambda(\tau) d\tau \right)$$

where $\lambda_p(t) = \lambda_{k_p}^{u_p, v_p}(t)$, and to represent the total survival probability for un-happened events we use: ¹

$$\Lambda(\tau) = \sum_{u=1}^n \sum_{v=1}^n \sum_{k \in \{0,1\}} \lambda_k^{u,v}(\tau)$$

¹In practice, mini-batches are applied (see their Appendix).

Algorithm 1 Update Algorithm for \mathcal{S} and \mathbf{A}

Input: Event record $o = (u, v, t, k)$, Event Intensity $\lambda_k^{u,v}(t)$ computed in (1), most recently updated $\mathbf{A}(\bar{t})$ and $\mathcal{S}(\bar{t})$. **Output:** $\mathbf{A}(t)$ and $\mathcal{S}(t)$

1. Update \mathbf{A} : $\mathbf{A}(t) = \mathbf{A}(\bar{t})$

if $k = 0$ then $\mathbf{A}_{uv}(t) = \mathbf{A}_{vu}(t) = 1$

$\leftarrow \{\text{Association event}\}$

2. Update \mathcal{S} : $\mathcal{S}(t) = \mathcal{S}(\bar{t})$

if $k = 1$ and $\mathbf{A}_{uv}(t) = 0$ return $\mathcal{S}(t), \mathbf{A}(t)$

$\leftarrow \{\text{Communication event, no Association exists}\}$

for $j \in \{u, v\}$ do

$b = \frac{1}{|\mathcal{N}_j(t)|}$ where $|\mathcal{N}_j(t)|$ is the size of $\mathcal{N}_j(t) = \{i : \mathbf{A}_{ij}(t) = 1\}$

$\mathbf{y} \leftarrow \mathcal{S}_j(t)$

if $k = 1$ and $\mathbf{A}_{uv}(t) = 1$ then {

$\leftarrow \{\text{Communication event, Association exists}\}$

$\mathbf{y}_i = b + \lambda_k^{ji}(t)$ where i is the other node involved in the event.

$\leftarrow \{\lambda \text{ computed in Eq. 2}\}$

else if $k = 0$ and $\mathbf{A}_{uv}(t) = 0$ then {

$\leftarrow \{\text{Association event}\}$

$b' = \frac{1}{|\mathcal{N}_j(\bar{t})|}$ where $|\mathcal{N}_j(\bar{t})|$ is the size of $\mathcal{N}_j(\bar{t}) = \{i : \mathbf{A}_{ij}(\bar{t}) = 1\}$

$x = b' - b$

$\mathbf{y}_i = b + \lambda_k^{ji}(t)$ where i is the other node involved in the event

$\leftarrow \{\lambda \text{ computed in Eq. 2}\}$

$\mathbf{y}_w = \mathbf{y}_w - x; \forall w \neq i, \mathbf{y}_w \neq 0$

end if

Normalize \mathbf{y} and set $\mathcal{S}_j(t) \leftarrow \mathbf{y}$

end for

return $\mathcal{S}(t), \mathbf{A}(t)$

DyRep experiments focus on the dynamic feature of the model.

- **Dynamic Link Prediction:** given v, k, t fixed, which is the most likely u ?

$$f_k^{u,v}(t) = \lambda_k^{u,v}(t) \exp \left(\int_{\bar{t}}^t \lambda(s) ds \right)$$

is the conditional density used to find the most likely node, where \bar{t} is the time of the most recent event on u or v .

- **Event Time Prediction:** what is the next time point when a particular type of event occur?

$$\hat{t} = \int_t^\infty t f_k^{u,v}(t) dt$$

A dynamic graph \mathbb{G} is defined as a series of observed static graph snapshots:

$$\mathbb{G} = \{\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^T\}$$

where each snapshot \mathcal{G}^t is defined as:

$$\mathcal{G}^t = (\mathcal{V}, \mathcal{E}^t)$$

it is a **weighted undirected** graph with a shared node set \mathcal{V} . The corresponding weighted adjacency matrix at time t is \mathcal{A}^t .

Idea: to learn $\mathbf{e}_v^t \in \mathbb{R}^d$, the node representations, preserving (1) the local graph structures centered at v , (2) its temporal evolutionary behaviors at time t (e.g. link connection and removal)

Self-attention mechanism used in DySAT:

► Structural:

- At each \mathcal{G}^t ($t = 1, 2, \dots, T$)
- Exactly the same as what a standard GAT does (link)

$$\mathbf{z}_v = \sigma \left(\sum_{u \in \mathcal{N}_v} \alpha_{uv} \mathbf{W}^s \mathbf{x}_u \right)$$

where \mathbf{W}^s is shared by all nodes, attention weight α_{uv} is computed upon $\mathbf{W}^s \mathbf{x}_u$.

► Temporal:

- Over the sequence $\mathbb{G} = \{\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^T\}$

$$\mathbf{Z}_v = \beta_v(\mathbf{X}_v \mathbf{W}_v)$$

this time, attention weight $\beta_v \in \mathbb{R}^{T \times T}$ is computed upon $\mathbf{X}_v \mathbf{W}_q$, $\mathbf{X}_v \mathbf{W}_k$ and $\mathbf{M} \in \mathbb{R}^{T \times T}$.

We define $\mathbf{M} \in \mathbb{R}^{T \times T}$ as:²

$$M_{ij} = \begin{cases} 0 & i \leq j \\ -\infty & \text{otherwise} \end{cases}$$

The linear projection matrices to generate queries, keys, and values: $\mathbf{W}_k, \mathbf{W}_q, \mathbf{W}_v \in \mathbb{R}^{D' \times F'}$. $\beta_v \in \mathbb{R}^{T \times T}$ is computed as:

$$\beta_v^{ij} = \frac{\exp(e_v^{ij})}{\sum_{k=1}^T \exp e_v^{ik}}$$

where $e_v^{ij} \in \mathbb{R}$ is computed as:

$$e_v^{ij} = \left(\frac{((\mathbf{X}_v \mathbf{W}_q)(\mathbf{X}_v \mathbf{W}_k)^T)_{ij}}{\sqrt{F'}} + M_{ij} \right)$$

² \mathbf{M} forces the model to attend to previous time steps only.

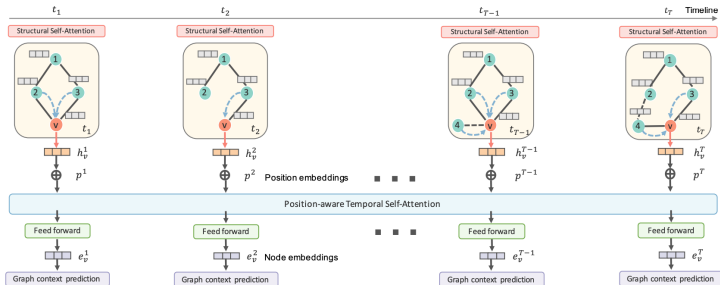


Figure: Multi-Faceted Graph Evolution is modeled by applying multiple attention heads to both structural and temporal attention.

Designed for: preserving the local structure around a node across multiple time steps.

$$L = \sum_{t=1}^T \sum_{v \in \mathcal{V}} \left(\sum_{u \in \mathcal{N}_{walk}^t(v)} -\log \sigma \langle \mathbf{e}_u^t, \mathbf{e}_v^t \rangle \right. \\ \left. - w_n \sum_{u' \in P_n^t(v)} \log(1 - \sigma \langle \mathbf{e}_{u'}^t, \mathbf{e}_v^t \rangle) \right)$$

Intuition: binary cross-entropy loss at each time step, with negative sampling, to encourage close ³ nodes to have similar representations.

³Close nodes are co-occurring in fixed-length random walks.

The experiments are focusing on link prediction.

- ▶ Single and multiple step link-prediction performances
- ▶ Link-prediction involving unseen nodes and links
- ▶ Ablation studies on the attention layers

Significantly outperforms the SOTA models, and found that within the range of $(1, 16)$, the more attention heads, the better. More findings are in their paper.

A dynamic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is **directed** in this case, and it involves N nodes:

$$\mathcal{V} = \{v_1, v_2, \dots, v_N\}$$

and a directed edge e could be represented as (v_s, v_g, t) , meaning an edge linked from v_s to v_g at time t .

This time, “right before time t ” is denoted as $t-$.

Idea: to learn an embedding, dynamic is achieved by the Update and Propagation components working together.

There are two major components of the model:

- ▶ **Update Component:** based on the long-short term memory (LSTM) unit.
- ▶ **Propagation Component:** very similar with a standard GAT layer's propagation, except some details e.g. the selection of activation function, and:
 - ▶ ignoring the very-old neighbors (long-time no interaction) that hasn't interacted for an interval of $\Delta > \tau$.

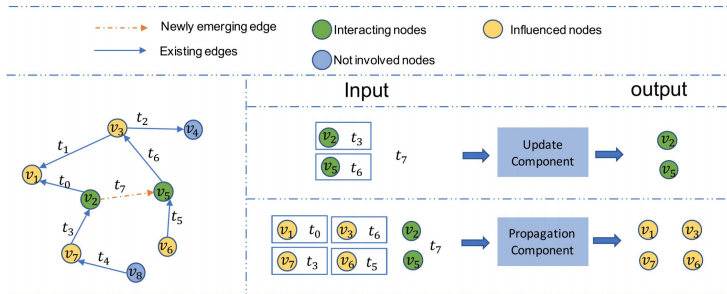


Figure: What happened in DyGNN when a new interaction happened at t_7 from v_2 to v_5 . v_1, v_3, v_6, v_7 , the direct neighbors, are the influenced nodes. For more details please refer to their paper.

The DyGNN model itself serves as an encoder that gives $\mathbf{u}_v(t)$ as the node v 's embedding at time t .

Losses are different depending on different downstream tasks in the decoder.

- ▶ Link Prediction: negative log-sigmoid of the source & target inner product; negative-sampling used.
- ▶ Node Classification: cross-entropy loss at the last layer (unit = 2).

On a dynamic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (can be **directed** or **undirected**), all interactions ($e \in \mathcal{E}$) have time associated with them.

We seek to learn a continuous functional mapping $\Phi : T \rightarrow \mathbb{R}^{d_T}$ to encode time, where time domain $T = [0, t_{max}]$ (t_{max} is determined by the observed data).

Idea: learn time-aware embedding, using functional time encoding and the temporal graph attention layer (TGAT layer).

- ▶ TGAT layer: a local aggregation operator that takes (1) the temporal neighborhood with their hidden representations (or features) and (2) timestamps as input, and the output is the time-aware representation.

For node v_0 at time t , we define its neighborhood as:

$$\mathcal{N}(v_0; t) = \{v_1, v_2, \dots, v_N\}$$

where, for each $v_i \in \mathcal{N}(v_0; t)$, the interaction between v_0 and v_i took place at $t_i < t$.

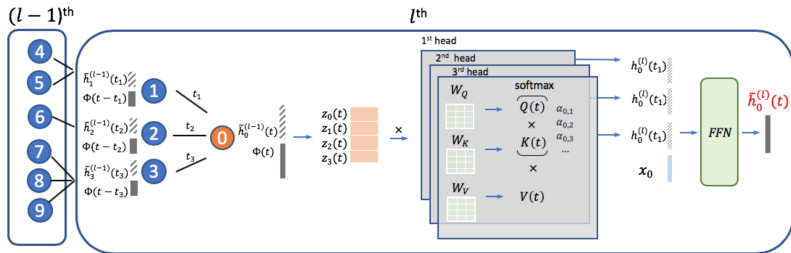


Figure: The architect of the l^{th} TGAT layer with $k = 3$ attention heads for node v_0 at time t . Output is $\tilde{\mathbf{h}}_i^{(l)}(t)$ where $i = 0$ is the node index. Feature vectors $\tilde{\mathbf{h}}_i^{(l-1)}(t)$ and $\Phi(t - t_i)$ are simply concatenated, as the layer's input. $\Phi(t - t_i) \in \mathbb{R}^{d_T}$ takes the place of positional encoding in a standard transformer layer (ref). The remaining parts (masked multi-head self attention etc.) are almost the same as GAT.

It seems that there's a glitch in their paper writing (if you follow their description of Φ_d , the output dimension will be $2d$ instead of d), the Φ_d implemented in the code is:

$$\Phi_d(t) = [\cos(\omega_1 t + \theta_1), \cos(\omega_2 t + \theta_2), \dots, \cos(\omega_d t + \theta_d)] \in \mathbb{R}^d$$

where both $\boldsymbol{\omega} = [\omega_1, \dots, \omega_d]$ and $\boldsymbol{\theta} = [\theta_1, \dots, \theta_d]$ are parameters to be trained.

In fact, we should consider $\Phi(t_i)$ instead of $\Phi(t - t_i)$ ($i = 1, 2, \dots, N$). However, we are only interested in the timespan:

$$|t_i - t_j| = |(t - t_i) - (t - t_j)|$$

so it doesn't matter which way we use it.

Two kinds of tasks:

- ▶ **Transductive task:** node classification & link prediction on **observed** nodes.
- ▶ **Inductive task:** node classification & link prediction involving **unseen** nodes.

It outperforms all SOTA models under all tasks.

1. No agreement at all on how to model dynamic graph.
2. Multi-head self-attention mechanism is very frequently applied.
3. To model a continuous time stream, people usually define a *continuous function* and learn its parameters.
4. etc.