中国科学院大学
University of Chinese Academy of Sciences

Representative Learning on Dynamic Graphs: A Survey

Encoder and Decoder of Dynamic Graph

UCAS

He Yinhan

06/11/2021

干啥?

社会关系等复杂关系网络可以用**图**表示出来，那种边儿内含有特殊信息的图就是**知识图谱**

而这种关系往往是**动态**的。假设咱手上有些信息，比如

贺胤涵 金泰妍 粉丝见面 2020.11.2
刁若愚 侯建国 吃饭 2021.09.6
....（省略100000000000条2010-2021.11.6发生的事儿)
王一佳 羽生结弦 第二次看电影儿 2021.04.6

那你能**预测（extrapolation）**

贺胤涵会在2022.04.8和金泰妍一起参加啥活动? -----对关系预测
刁若愚会在2022.04.9和谁吃饭? -----对节点预测
王一佳会在啥时候再跟羽生结弦看电影儿? -----对时间预测

干啥?

现实中，知识库不可能是完备的，对于知识库的**时间跨度内**的但没在知识库里的事儿

你能**估测（Interpolation）**

贺胤涵在2018.04.8和金泰妍一起参加了啥活动?　-----对关系估测
刁若愚在2016.04.9和谁吃了饭?　-----对节点估测
王一佳在啥时候第一次跟羽生结弦看了电影儿?　-----对时间估测

干啥?

研究

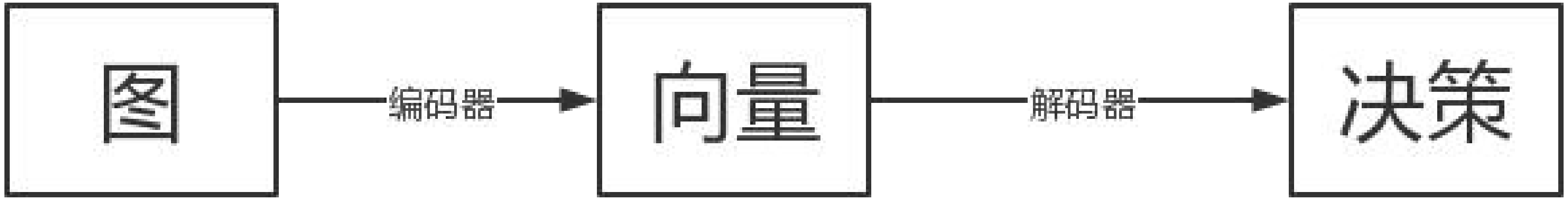
咋估，咋预测

BUT

这听起来像魔法一样激动人心，但是我们这次大都只会考虑边中**不存储特殊信息**的情况，也就是第二张幻灯片中**没有蓝色字**的情况

事实上，迄今为止，动态知识图谱预测**依然没有**很好的模型能给出令人**满意**的结果

咋弄?



图

编码器

向量

解码器

决策

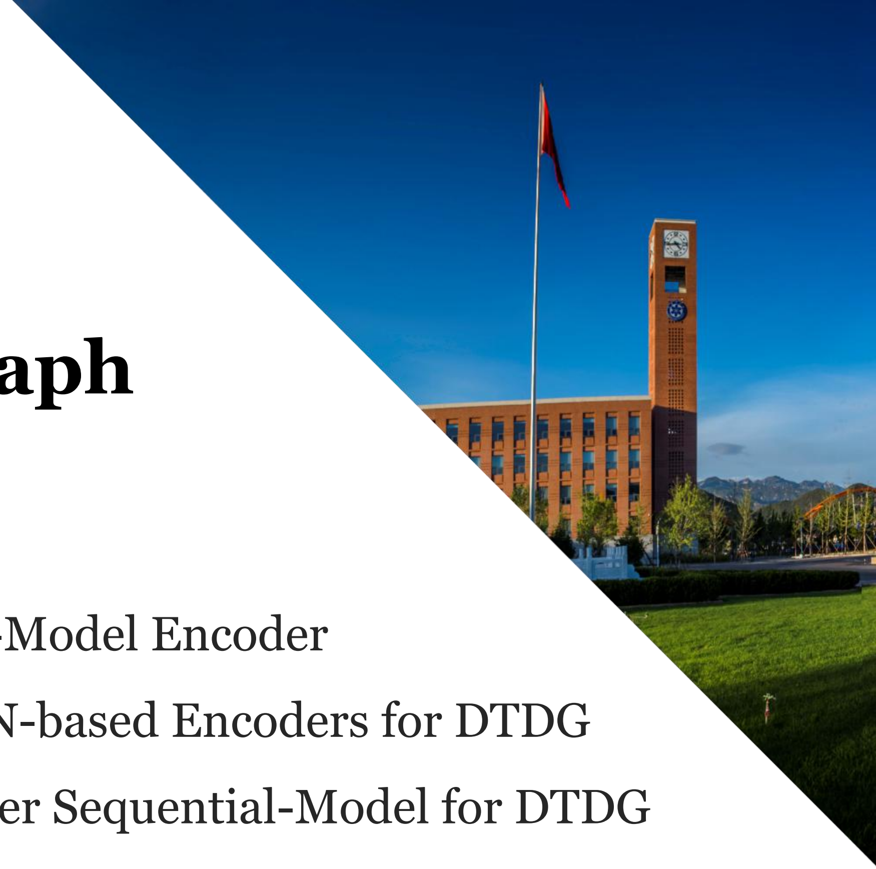
FRAME WORK

1. Encoder of Dynamic Graph

2. Decoder of Dynamic Graph

3. Future Work and Open Questions

4. Complementary Explanation for Graph



/01

1. Encoder of Dynamic Graph

1.1 Aggragating Temporal Observation

1.2 Aggregate Static Feature

1.3 Time as Regularizer

1.4 Decomposition-based Encoder

1.5 Random Walk Encoder

1.6 Sequence-Model Encoder

- 1.6.1 RNN-based Encoders for DTDG
- 1.6.2 Other Sequential-Model for DTDG
- 1.6.3 RNN-based Encoders for CTDG
- 1.6.4 Discussion and Analysis

1.7 Autoencoder-based Encoder

1.8 Diachronic(历时的) Encoder

1.9 Staleness

Encoder for Dynamic Graph

For **interp** and **extrap** problems, although there are some encoders fit for both senarios, but usually inter and extra use different types of encoders.

Extrap: needs encoder that provide node and relation embeddings based only on observations **in the past**

Interp: needs encoder that provide node and relation embeddings based on observations **before, at, and after** time t.

1.1 Aggregation Temporal Observation

Idea is **collapsing the dynamic graph into a static graph** by aggregating temporal observations over time.

Ignore timestamps, simple aggregation way is

$$A_{sum}[i][j] = \sum_{t=1}^{T} A^t[i][j]$$

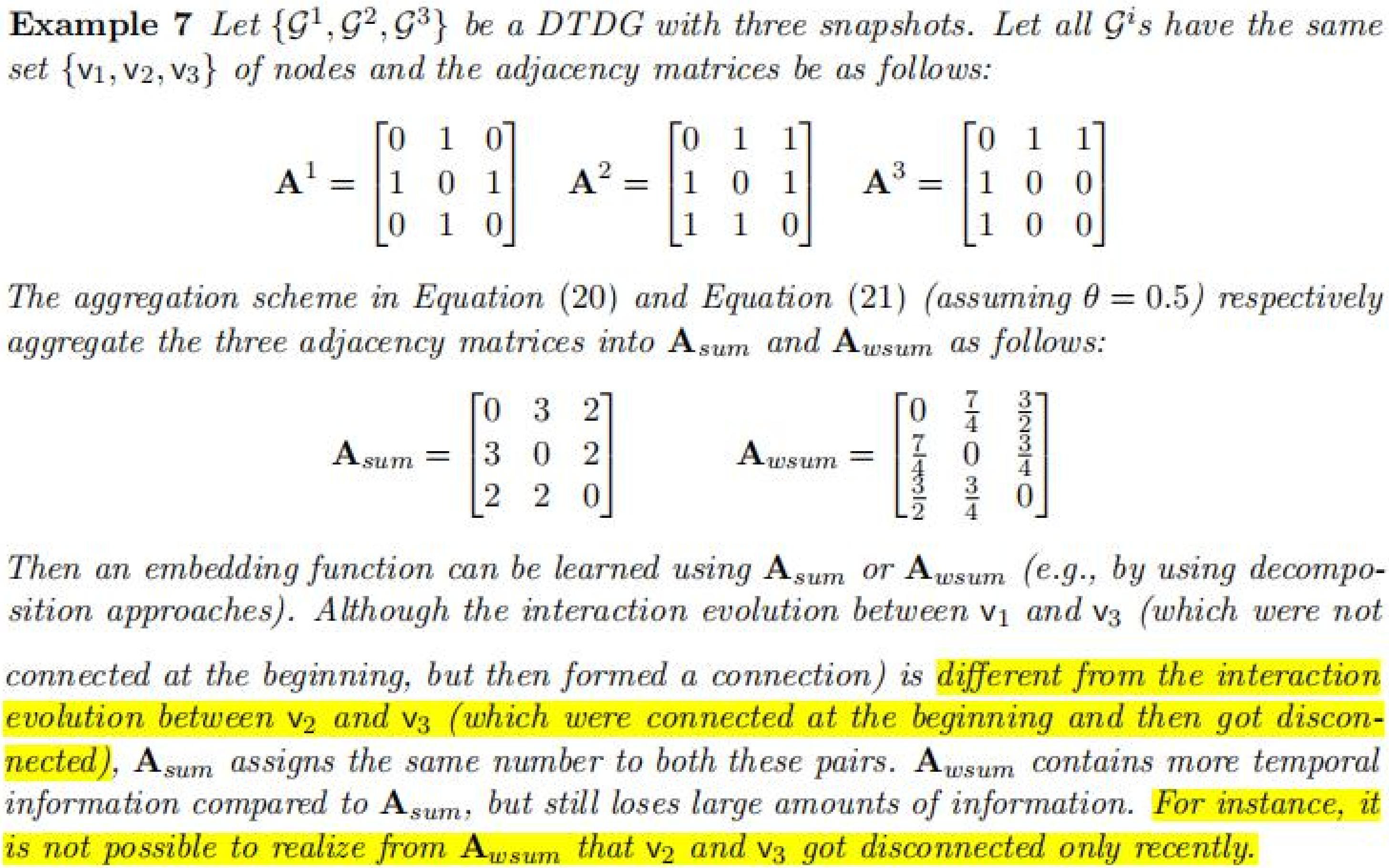
to add some time info, use

$$A_{wsum}[i][j] = \sum_{t=1}^{T} \theta^{T-t} A^t[i][j]$$

where $0 < \theta < 1$.Larger value for θ put more emphasis on **recent ad** matrix.

This method may **lose large amount of useful info** hindering them from making accurate predictions in many scenarios.

1.1 Aggregation Temporal Observation



Example 7 *Let $\{\mathcal{G}^1, \mathcal{G}^2, \mathcal{G}^3\}$ be a DTDG with three snapshots. Let all $\mathcal{G}^i$s have the same set $\{\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3\}$ of nodes and the adjacency matrices be as follows:*

$$\mathbf{A}^1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \mathbf{A}^2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \quad \mathbf{A}^3 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

The aggregation scheme in Equation (20) *and Equation* (21) *(assuming $\theta = 0.5$) respectively aggregate the three adjacency matrices into $\mathbf{A}_{sum}$ and $\mathbf{A}_{wsum}$ as follows:*

$$\mathbf{A}_{sum} = \begin{bmatrix} 0 & 3 & 2 \\ 3 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \qquad \mathbf{A}_{wsum} = \begin{bmatrix} 0 & \frac{7}{4} & \frac{3}{2} \\ \frac{7}{4} & 0 & \frac{3}{4} \\ \frac{3}{2} & \frac{3}{4} & 0 \end{bmatrix}$$

Then an embedding function can be learned using $\mathbf{A}_{sum}$ or $\mathbf{A}_{wsum}$ (e.g., by using decomposition approaches). Although the interaction evolution between v_1 and v_3 (which were not connected at the beginning, but then formed a connection) is different from the interaction evolution between v_2 and v_3 (which were connected at the beginning and then got disconnected), $\mathbf{A}_{sum}$ assigns the same number to both these pairs. $\mathbf{A}_{wsum}$ contains more temporal information compared to $\mathbf{A}_{sum}$, but still loses large amounts of information. For instance, it is not possible to realize from $\mathbf{A}_{wsum}$ that v_2 and v_3 got disconnected only recently.

1.2 Aggregating Static Features

Procedure:

1. apply a **static encoder** to each snapshot.
2. **aggregate the results** over time.

Computationally Expensive!!

Example:

Yao (2016) aggregate features into a **single feature vector** as follows:

$$z_v = \sum_{t=1}^{T} exp(-\theta(T-t))z_v^t$$

which exponentially decaying older feature.

Rather than explicitly defined aggregator that assigns prefixed weights to previous snpshots, one can fit **time series model** to features from previous snapshots to use this model to **predict values of this feature** for the next snapshot. **Gunes(2016) used ARIMA**, Moradabadi(2017) use **approach based on reinforcement learning.**

1.2 Time as a regularizer

离散时间动态图（discrete time dynamic graph）

for DTDG:use time as a regularizer to impose a smooth constraint on embeding of each node over consecutive snapshots.

Consider a DTDG as {G1, . . . , GT }, let $EMB^{t-1}(v) = (z_v^{t-1})$ as embedding at (t-1) th snapshot. This approach uses static embeding but add one more constraint to calculate G^t. The constraint is to minimize

$$dist(z_v^t, z_v^{t-1}) = ||z_v^t - z_v^{t-1}||$$

Singer added a rotation matrix R^t, then

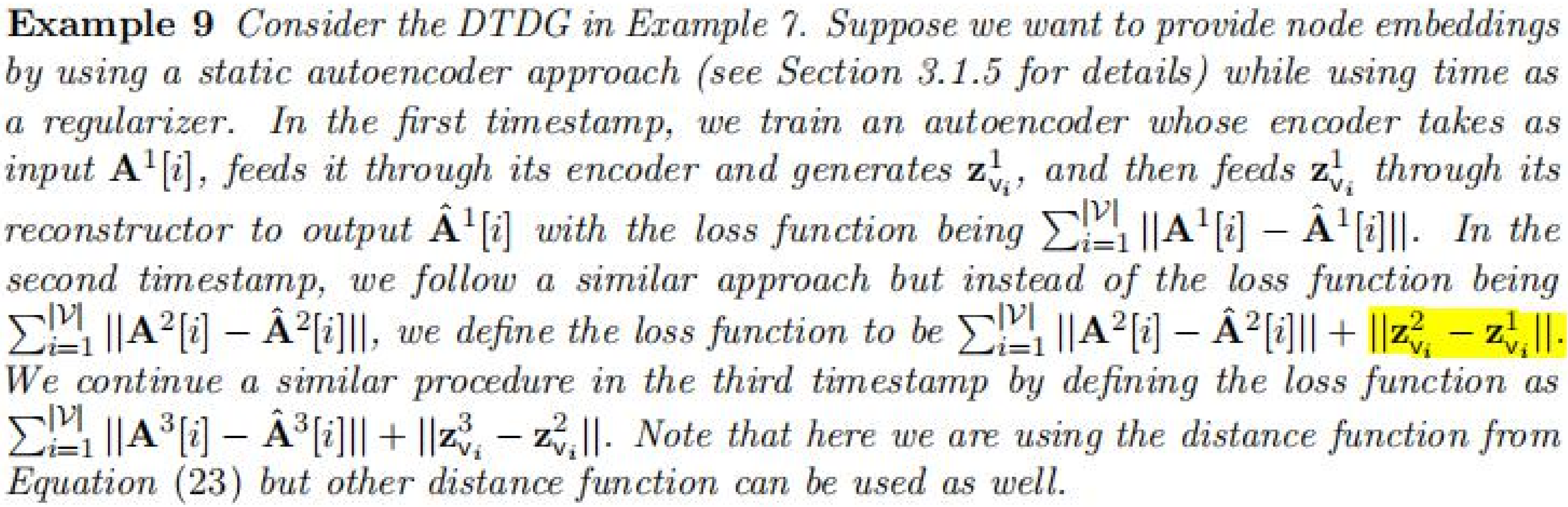
$$dist(z_v^t, z_v^{t-1}) = ||R^t z_v^t - z_v^{t-1}||$$

some use other than Euclidean distance

$$dist(z_v^t, z_v^{t-1}) = 1 - (z_v^t)' z_v^{t-1}$$

where all embeding vectors are restricted to have norm of 1

1.2 Time as a regularizer



Example 9 *Consider the DTDG in Example 7. Suppose we want to provide node embeddings by using a static autoencoder approach (see Section 3.1.5 for details) while using time as a regularizer. In the first timestamp, we train an autoencoder whose encoder takes as input* $\mathbf{A}^1[i]$, *feeds it through its encoder and generates* $\mathbf{z}^1_{\mathrm{v}_i}$, *and then feeds* $\mathbf{z}^1_{\mathrm{v}_i}$ *through its reconstructor to output* $\hat{\mathbf{A}}^1[i]$ *with the loss function being* $\sum_{i=1}^{|\mathcal{V}|} ||\mathbf{A}^1[i] - \hat{\mathbf{A}}^1[i]||$. *In the second timestamp, we follow a similar approach but instead of the loss function being* $\sum_{i=1}^{|\mathcal{V}|} ||\mathbf{A}^2[i] - \hat{\mathbf{A}}^2[i]||$, *we define the loss function to be* $\sum_{i=1}^{|\mathcal{V}|} ||\mathbf{A}^2[i] - \hat{\mathbf{A}}^2[i]|| + ||\mathbf{z}^2_{\mathrm{v}_i} - \mathbf{z}^1_{\mathrm{v}_i}||$. *We continue a similar procedure in the third timestamp by defining the loss function as* $\sum_{i=1}^{|\mathcal{V}|} ||\mathbf{A}^3[i] - \hat{\mathbf{A}}^3[i]|| + ||\mathbf{z}^3_{\mathrm{v}_i} - \mathbf{z}^2_{\mathrm{v}_i}||$. *Note that here we are using the distance function from Equation (23) but other distance function can be used as well.*

1.3 Time as a regularizer

Imposing **smoothness constraints** through penalizing distance between vector representations of a node at consecutive snapshots **stops vector representations having sharp changes.**

But, In some conditions, a node may **change substantially** from one snapshot to the other. For example, if a company gets acquired by a large company, its vector representation at the next snapshot would **sharply change**. Instead, one may initialize representations for time t with the learned represenations at time $t-1$ and **allow static encoder to further optimize the representation at time t**. This procedure implicitly impose smoothness constraints while also allow sharp changes when neccessary. (Goyal 2017)

Another notable “time as regularizer” model is an extension of LINE, which is a static graph model. Besides using time as regularizer, they propose a way of **recomputing the node embedings only for nodes that have been influenced greatly from the last snapshot.(2018)**

How Goyal deal with sharp changes

Method they take:

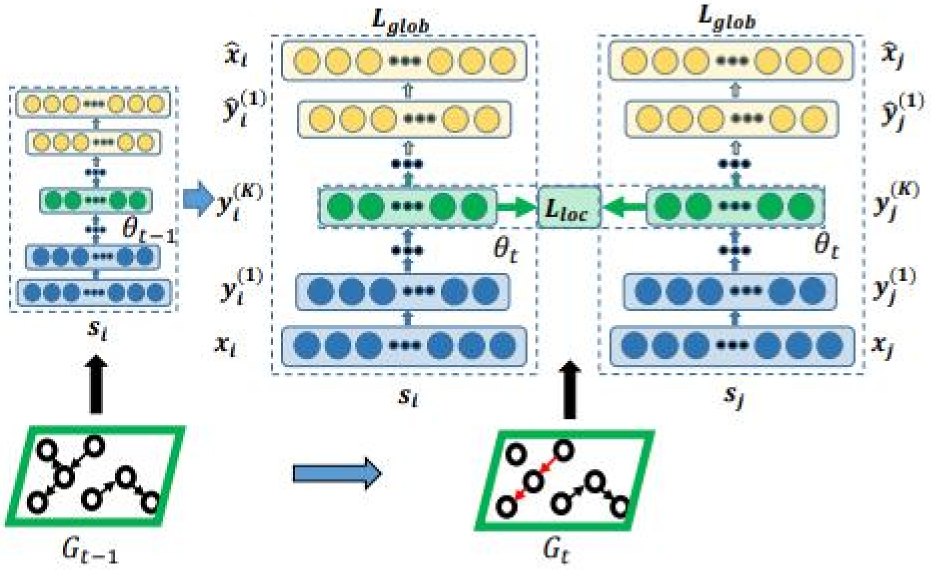
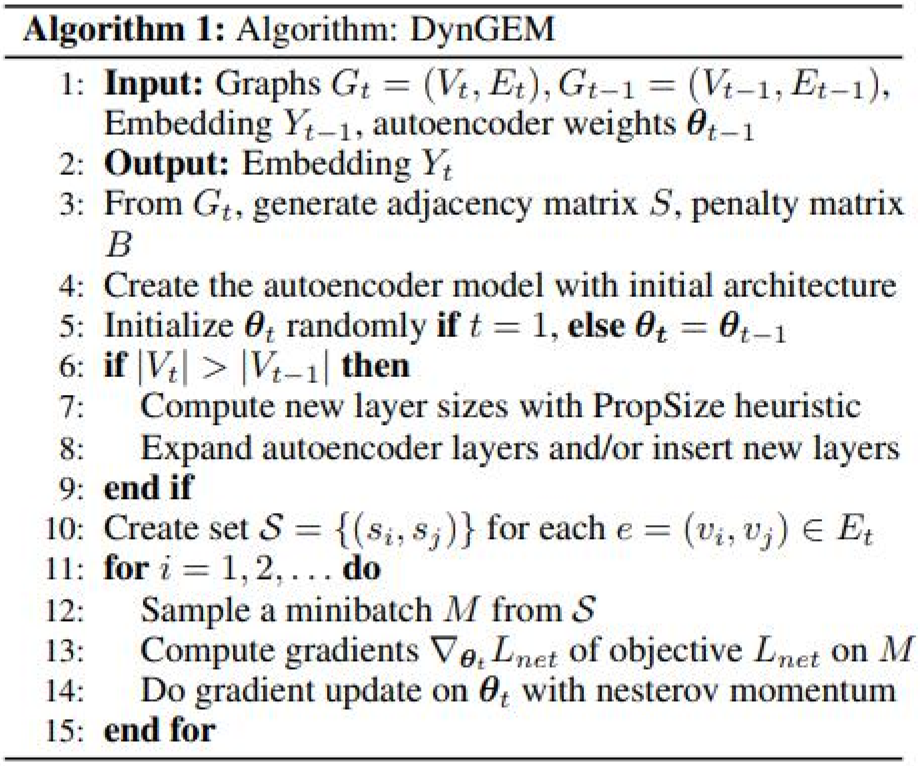


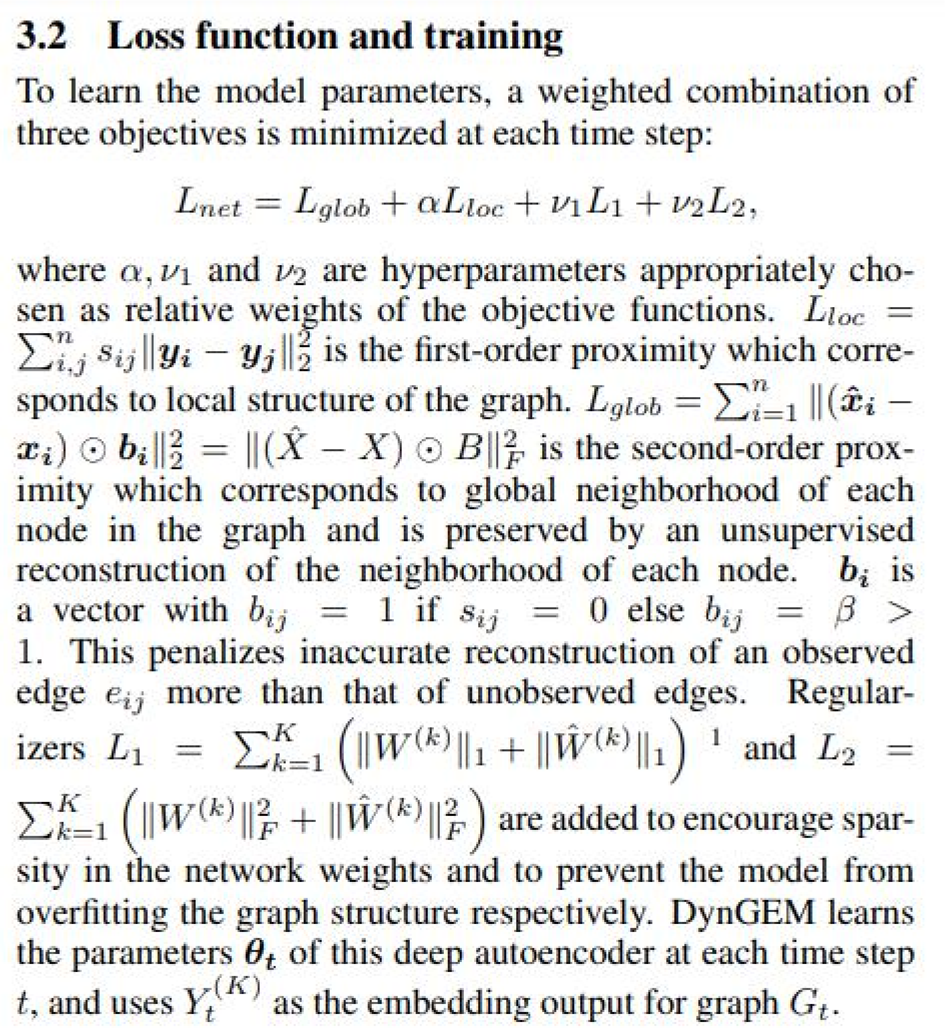
Figure 1: DynGEM: Dynamic Graph Embedding Model. The figure shows two snapshots of a dynamic graph and the corresponding deep autoencoder at each step.



Algorithm 1: Algorithm: DynGEM

```
Input: Graphs $G_t = (V_t, E_t), G_{t-1} = (V_{t-1}, E_{t-1})$, Embedding $Y_{t-1}$, autoencoder weights $\theta_{t-1}$
Output: Embedding $Y_t$
From $G_t$, generate adjacency matrix $S$, penalty matrix $B$
Create the autoencoder model with initial architecture
Initialize $\theta_t$ randomly if $t = 1$, else $\theta_t = \theta_{t-1}$
if $|V_t| > |V_{t-1}|$ then
    Compute new layer sizes with PropSize heuristic
    Expand autoencoder layers and/or insert new layers
end if
Create set $\mathcal{S} = \{(s_i, s_j)\}$ for each $e = (v_i, v_j) \in E_t$
for $i = 1, 2, \ldots$ do
    Sample a minibatch $M$ from $\mathcal{S}$
    Compute gradients $\nabla_{\theta_t} L_{net}$ of objective $L_{net}$ on $M$
    Do gradient update on $\theta_t$ with nesterov momentum
end for
```

Why dealt?



3.2 Loss function and training

To learn the model parameters, a weighted combination of three objectives is minimized at each time step:

$$L_{net} = L_{glob} + \alpha L_{loc} + \nu_1 L_1 + \nu_2 L_2,$$

where α, ν_1 and ν_2 are hyperparameters appropriately chosen as relative weights of the objective functions. $L_{loc} = \sum_{i,j}^{n} s_{ij} \|\boldsymbol{y_i} - \boldsymbol{y_j}\|_2^2$ is the first-order proximity which corresponds to local structure of the graph. $L_{glob} = \sum_{i=1}^{n} \|(\boldsymbol{\hat{x}_i} - \boldsymbol{x_i}) \odot \boldsymbol{b_i}\|_2^2 = \|(\hat{X} - X) \odot B\|_F^2$ is the second-order proximity which corresponds to global neighborhood of each node in the graph and is preserved by an unsupervised reconstruction of the neighborhood of each node. $\boldsymbol{b_i}$ is a vector with $b_{ij} = 1$ if $s_{ij} = 0$ else $b_{ij} = \beta > 1$. This penalizes inaccurate reconstruction of an observed edge e_{ij} more than that of unobserved edges. Regularizers $L_1 = \sum_{k=1}^{K} \left(\|W^{(k)}\|_1 + \|\hat{W}^{(k)}\|_1\right)$ [1] and $L_2 = \sum_{k=1}^{K} \left(\|W^{(k)}\|_F^2 + \|\hat{W}^{(k)}\|_F^2\right)$ are added to encourage sparsity in the network weights and to prevent the model from overfitting the graph structure respectively. DynGEM learns the parameters $\boldsymbol{\theta_t}$ of this deep autoencoder at each time step t, and uses $Y_t^{(K)}$ as the embedding output for graph G_t.

- Because they did not use term
- $||EMB(v)_t - EMB(v)_{t-1}||$
- as regularizer, but for their using of θ_{t-1} to calculate θ_t, it also take graph evolution into account.

1.4 Decomposition based Encoders

Consider DTDG $\{G^1, G^2, \ldots, G^T\}$ **without change of vertex**, the adjacency matrices $A^1, A^2, \ldots, A^T$ for T timestamps can be **stacked into an order 3 tensor** $A \in R^{|V|\times|V|\times T}$. Then one can do d-component decomposition

$$A \approx \sum_{k=1}^{d} \lambda_k a_k \otimes b_k \otimes c_k$$

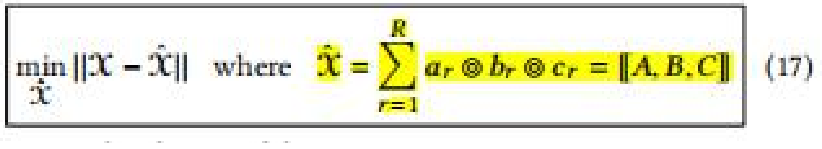
where $\lambda_k \in \mathbb{R}_+$, a_k,$b_k \in \mathbb{R}^{|V|}$,$c_k \in \mathbb{R}^T$, and temporal pattern is captured in $c_k s$. And a_k,b_k can be viewed as node embedding. The embedding can be used to **make prediction about time** $1 \leq t \leq T$, i.e. for interpolation.

More about tensor in ML:https://arxiv.org/pdf/1711.10781.pdf

Q: How to use c_k for inter. problem?

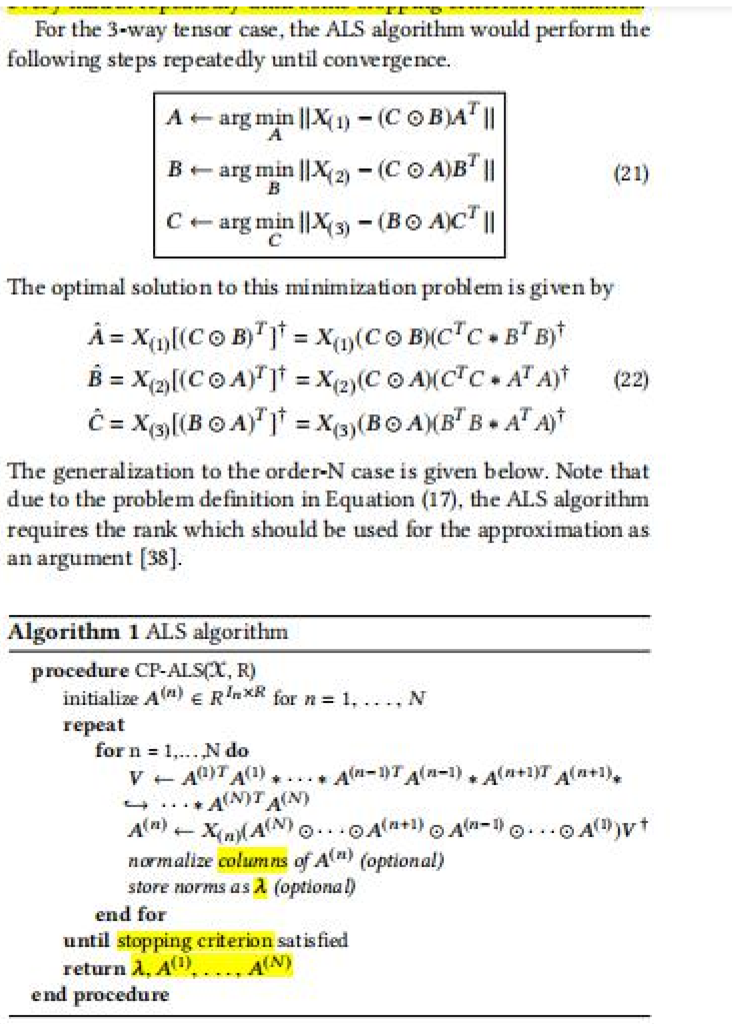
CP (canonical polyatic) decomposition

Question:



$$\min_{\hat{\mathcal{X}}} \|\mathcal{X} - \hat{\mathcal{X}}\| \quad \text{where} \quad \hat{\mathcal{X}} = \sum_{r=1}^{R} a_r \circledcirc b_r \circledcirc c_r = [\![A, B, C]\!] \qquad (17)$$

Solution:



For the 3-way tensor case, the ALS algorithm would perform the following steps repeatedly until convergence.

$$\boxed{\begin{aligned} A &\leftarrow \arg\min_{A} \|X_{(1)} - (C \odot B)A^T\| \\ B &\leftarrow \arg\min_{B} \|X_{(2)} - (C \odot A)B^T\| \\ C &\leftarrow \arg\min_{C} \|X_{(3)} - (B \odot A)C^T\| \end{aligned}} \qquad (21)$$

The optimal solution to this minimization problem is given by

$$\begin{aligned} \hat{A} &= X_{(1)}[(C \odot B)^T]^\dagger = X_{(1)}(C \odot B)(C^T C * B^T B)^\dagger \\ \hat{B} &= X_{(2)}[(C \odot A)^T]^\dagger = X_{(2)}(C \odot A)(C^T C * A^T A)^\dagger \\ \hat{C} &= X_{(3)}[(B \odot A)^T]^\dagger = X_{(3)}(B \odot A)(B^T B * A^T A)^\dagger \end{aligned} \qquad (22)$$

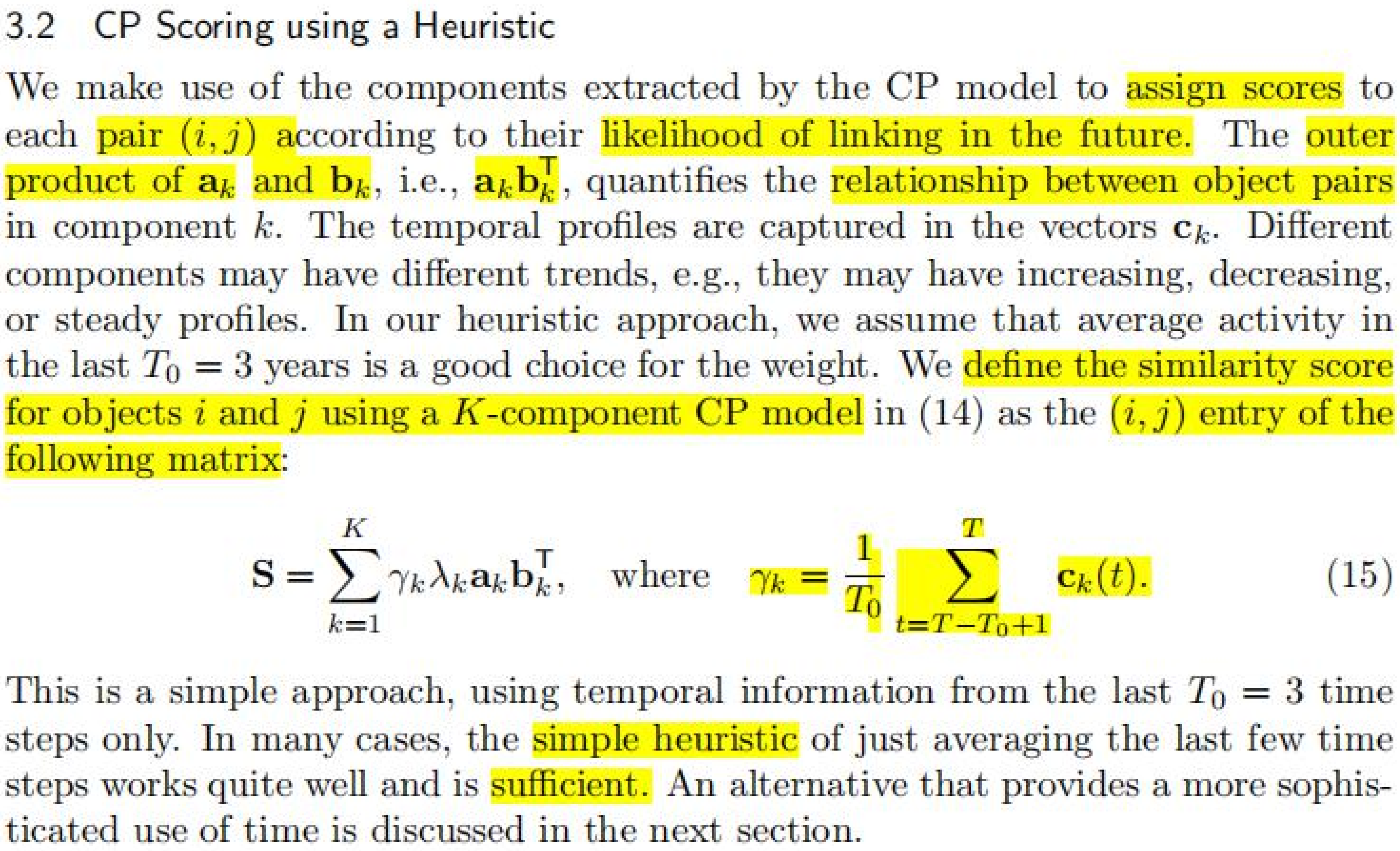
The generalization to the order-N case is given below. Note that due to the problem definition in Equation (17), the ALS algorithm requires the rank which should be used for the approximation as an argument [38].

Algorithm 1 ALS algorithm

```
procedure CP-ALS($\mathcal{X}$, R)
    initialize $A^{(n)} \in R^{I_n \times R}$ for $n = 1, \ldots, N$
    repeat
        for n = 1,...,N do
            $V \leftarrow A^{(1)T}A^{(1)} * \cdots * A^{(n-1)T}A^{(n-1)} * A^{(n+1)T}A^{(n+1)} *$
            $\hookrightarrow \cdots * A^{(N)T}A^{(N)}$
            $A^{(n)} \leftarrow X_{(n)}(A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \cdots \odot A^{(1)})V^\dagger$
            normalize columns of $A^{(n)}$ (optional)
            store norms as $\lambda$ (optional)
        end for
    until stopping criterion satisfied
    return $\lambda, A^{(1)}, \ldots, A^{(N)}$
end procedure
```

Attention: Algorithm is influenced by its initialization.

Using CP to define similarity score.



3.2 CP Scoring using a Heuristic

We make use of the components extracted by the CP model to assign scores to each pair (i, j) according to their likelihood of linking in the future. The outer product of $\mathbf{a}_k$ and $\mathbf{b}_k$, i.e., $\mathbf{a}_k\mathbf{b}_k^{\mathsf{T}}$, quantifies the relationship between object pairs in component k. The temporal profiles are captured in the vectors $\mathbf{c}_k$. Different components may have different trends, e.g., they may have increasing, decreasing, or steady profiles. In our heuristic approach, we assume that average activity in the last $T_0 = 3$ years is a good choice for the weight. We define the similarity score for objects i and j using a K-component CP model in (14) as the (i, j) entry of the following matrix:

$$\mathbf{S} = \sum_{k=1}^{K} \gamma_k \lambda_k \mathbf{a}_k \mathbf{b}_k^{\mathsf{T}}, \quad \text{where} \quad \gamma_k = \frac{1}{T_0} \sum_{t=T-T_0+1}^{T} \mathbf{c}_k(t). \qquad (15)$$

This is a simple approach, using temporal information from the last $T_0 = 3$ time steps only. In many cases, the simple heuristic of just averaging the last few time steps works quite well and is sufficient. An alternative that provides a more sophisticated use of time is discussed in the next section.

1.4 Decomposition based Encoders

For extrapolation, Dunlavy (2011) used **Halt-Winter methods** : given c_k, it predict a L dimensional vector $c_k^{'}$, which is the prediciton of the temporal factor of the next L timestamps. then just predict the adjacency matrix for the future L timestamps are

$$\widehat{A} \approx \sum_{k=1}^{d} \lambda_k a_k \otimes b_k \otimes c'_k$$

H-W method is a time series prediction method.

ATTENTION:After some time steps,one needs to **update the tensor decomposition** for more accurate future predictions. The recomputation can be **quite costly** so one can try **incremental updates**

1.4 Decomposition based Encoders

Yu(2017) use a way cooperating **temporal dependencies** into embeddings with decomposition methods. Given $A^1, \dots, A^T$, Yu predict $\widehat{A^{T+l}}$ as follow:

1. solve optimization problem:

$$min(\sum_{t=T-w}^{T} e^{-\theta(T-t)} ||A^t - U(V^t)'(P^t)'||_F^2)$$

where $P^t = (1-\alpha)(I - \alpha\sqrt{D^t}A^t\sqrt{D^t})^{-1}$ is **projection onto feature space** that neighboring nodes have similar embedings, ω is a **window of timestamps** into consideration, α ∈ (0, 1) is a regularization parameter, θ is a
decay parameter, $U \in \mathbb{R}^{|V|\times d}$ does not depend on time, and $V^t \in \mathbb{R}^{|V|\times d}$ is with **explicit time dependency**.V^t 是矩阵参数，以时间为变量的多项式

2. The prediction is gained as

$$\widehat{A}^{T+l} = U(V^{T+l})'$$

Q: Where are U,V from ?

From encoder-decoder view, U is seen as **static node feature** and $\mathrm{V^t}$s are time dependent features.(ith row is the embedding of ith node)

1.4 Decomposition based Encoders

Now extend tensor decomposition to case of temporal KGs by **modeling KG as order 4 tensor**

$$A \in \mathbb{R}^{|V| \times |R| \times |V| \times T}$$

and decomposing it to obtain entity, relation, and timestamp embeddings.

Streaming scenario for decomposition method

找节点的embedding实则是进行对

$$Ly = \lambda Dy$$

进行特征值分解，见文章p12

Recalculating decomposition at every timestamp is computationally expensive, use **incremental algorithm** to update current state in streaming case. **Incremental eigenvalue decomposition** (Chen(2015)) is based on perturbation theory. They use $\Delta L, \Delta D$ from G^T to G^{T+1} to compute the change of eigenvalue and eigenvectors, such as:

$$\Delta\lambda_i = \frac{y_i^T \Delta L y_i - \lambda_i y_i^T \Delta D y_i}{y_i^T D y_i}$$

Levin(2018) figured a method to update the embedding of **newly added nodes** and Nielsen(1978) studied **how SVD can be updated through time**.

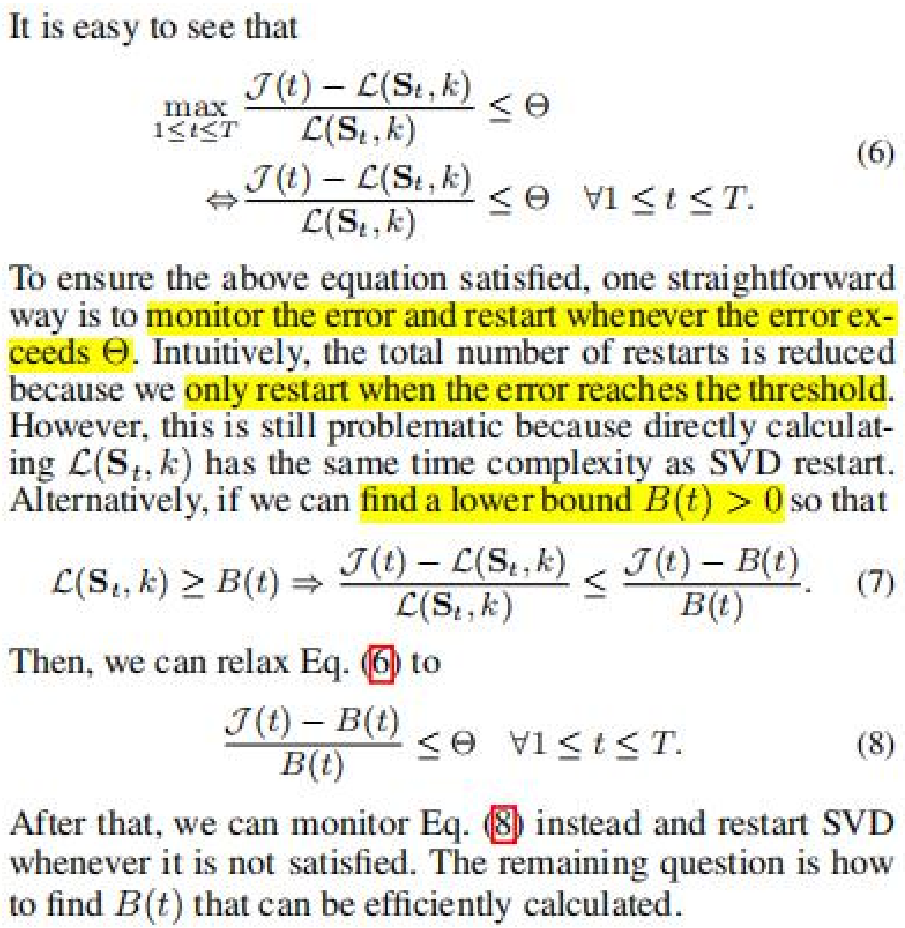
Streaming scenario for decomposition method

Problem: The approximation **error keeps accumulating** with incremental update, so one need to recaculate model from time to time. However, recauculation is expensive, need to **find proper time when error becomes tolerable**.

1. **Heuristic** methods: restart after a certain time.

2. Depend on graph dynamics. Zhang(2018) propose new method where given a tolerance threshold, it notifies at **what timestamp the approximation error exeeds threshold.**

How to monitor error?



It is easy to see that

$$
\begin{aligned}
&\max_{1\le t\le T} \frac{\mathcal{J}(t)-\mathcal{L}(\mathbf{S}_t,k)}{\mathcal{L}(\mathbf{S}_t,k)} \le \Theta \\
\Leftrightarrow &\frac{\mathcal{J}(t)-\mathcal{L}(\mathbf{S}_t,k)}{\mathcal{L}(\mathbf{S}_t,k)} \le \Theta \quad \forall 1 \le t \le T.
\end{aligned} \tag{6}
$$

To ensure the above equation satisfied, one straightforward way is to monitor the error and restart whenever the error exceeds Θ. Intuitively, the total number of restarts is reduced because we only restart when the error reaches the threshold. However, this is still problematic because directly calculating $\mathcal{L}(\mathbf{S}_t,k)$ has the same time complexity as SVD restart. Alternatively, if we can find a lower bound $B(t) > 0$ so that

$$
\mathcal{L}(\mathbf{S}_t,k) \ge B(t) \Rightarrow \frac{\mathcal{J}(t)-\mathcal{L}(\mathbf{S}_t,k)}{\mathcal{L}(\mathbf{S}_t,k)} \le \frac{\mathcal{J}(t)-B(t)}{B(t)}. \tag{7}
$$

Then, we can relax Eq. (6) to

$$
\frac{\mathcal{J}(t)-B(t)}{B(t)} \le \Theta \quad \forall 1 \le t \le T. \tag{8}
$$

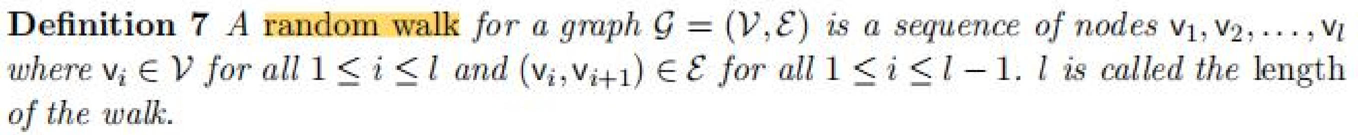
After that, we can monitor Eq. (8) instead and restart SVD whenever it is not satisfied. The remaining question is how to find $B(t)$ that can be efficiently calculated.

J(t) is the the reconstruction loss at time slice t, and the $L(S_t, k)$ is the minimum loss of SVD restart, so the subtraction means the effect of update error. θ is threshold.

1.5 Random Walk Encoders

Consider a DTDG $\{G^1, G^2, \ldots, G^T\}$, Mahdavi(2018)

1. generate random walk on G^1 like in static graph



Definition 7 *A random walk for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a sequence of nodes $v_1, v_2, \ldots, v_l$ where $v_i \in \mathcal{V}$ for all $1 \leq i \leq l$ and $(v_i, v_{i+1}) \in \mathcal{E}$ for all $1 \leq i \leq l-1$. l is called the length of the walk.*

2. feed those random walks to model M^1 to produce vector representation

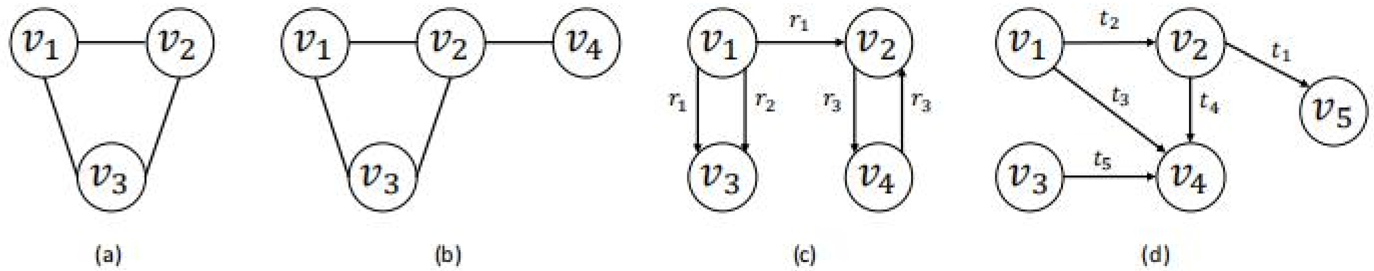
3. for t_th snapshot, keep valid random walk from (t-1)th snapshot, generate new random walk only starting at the **affected node**s.

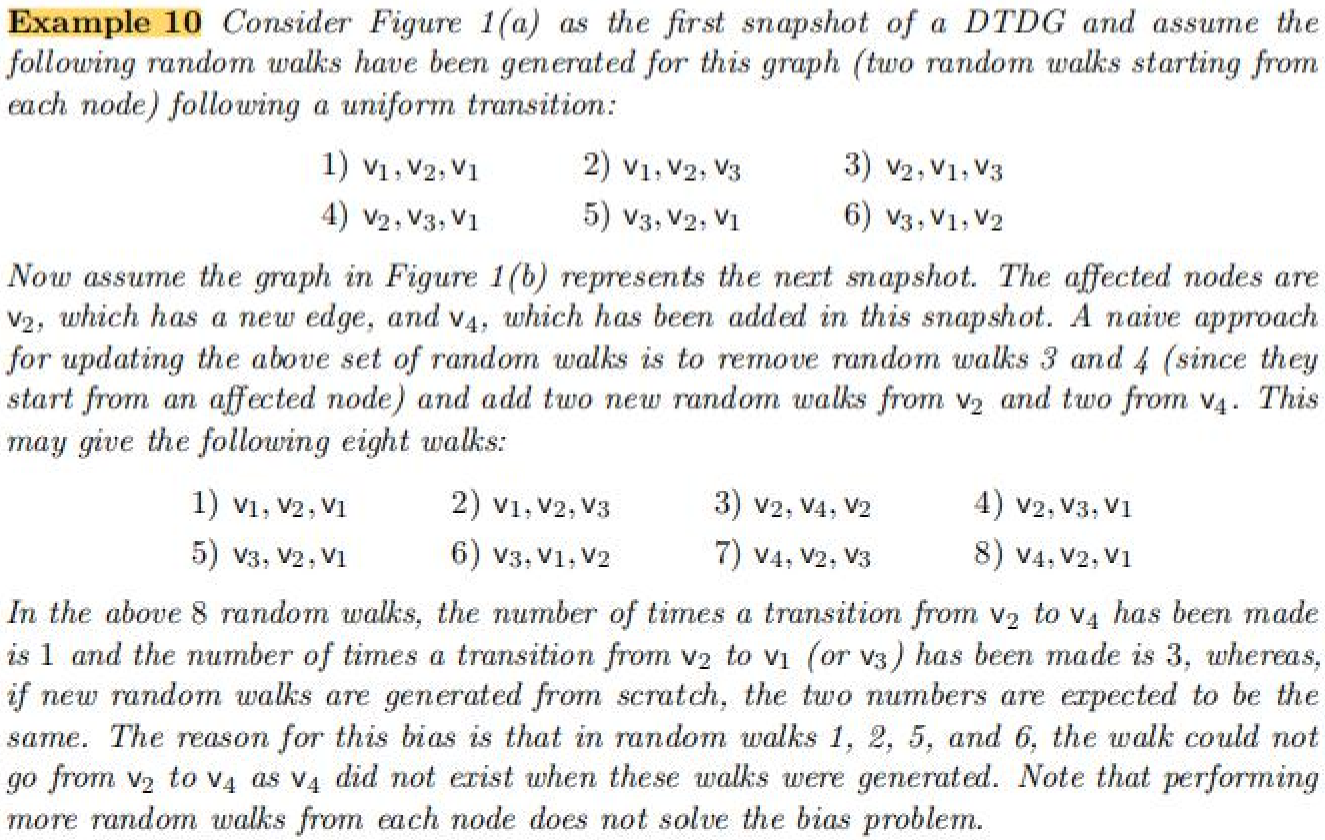
4. initialize M^t with learned parameters from M^{t-1} and allow M^t to be optimized and **produce embedding for t_th snapshot.**

Bian(2019) did similar thing in KG. They use **metapath2vec to generate random walk**s on initial KG, then at each snapshot, they use metapath2vec to generate random walks for affected nodes and recompute embeddings.

But, this method would **lead to bias**.

1.5 Random Walk Encoders





Example 10 *Consider Figure 1(a) as the first snapshot of a DTDG and assume the following random walks have been generated for this graph (two random walks starting from each node) following a uniform transition:*

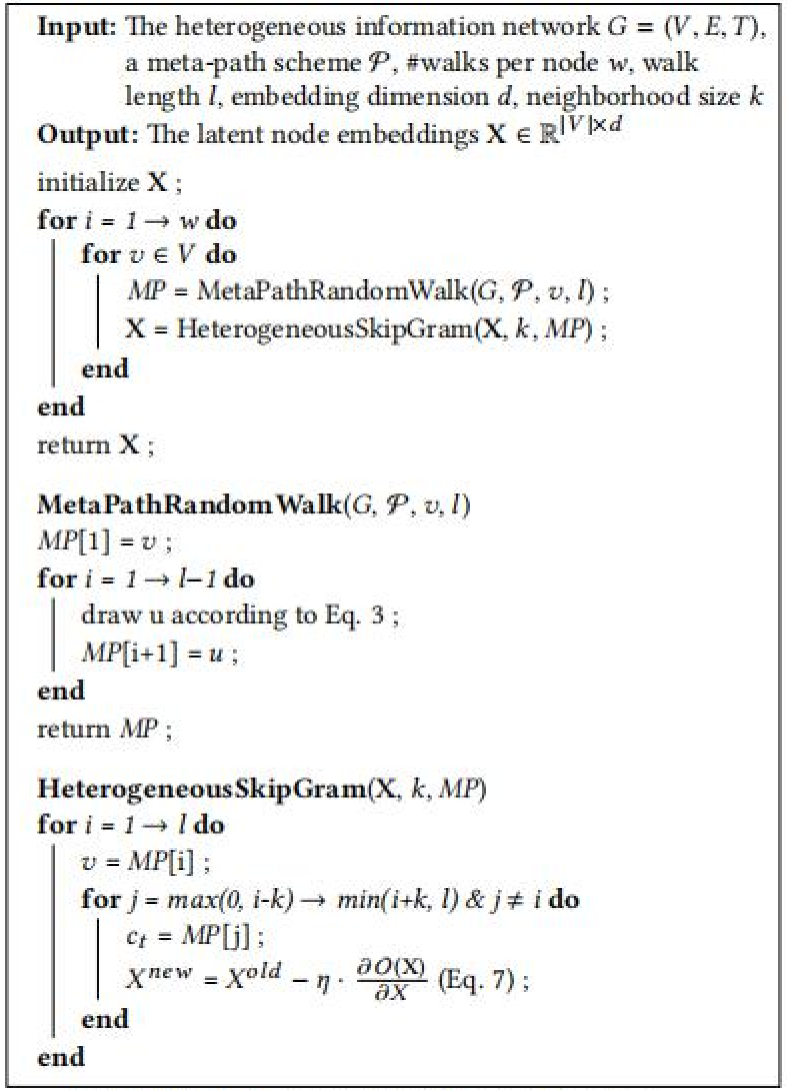
1) v_1, v_2, v_1 2) v_1, v_2, v_3 3) v_2, v_1, v_3
4) v_2, v_3, v_1 5) v_3, v_2, v_1 6) v_3, v_1, v_2

Now assume the graph in Figure 1(b) represents the next snapshot. The affected nodes are v_2*, which has a new edge, and* v_4*, which has been added in this snapshot. A naive approach for updating the above set of random walks is to remove random walks 3 and 4 (since they start from an affected node) and add two new random walks from* v_2 *and two from* v_4*. This may give the following eight walks:*

1) v_1, v_2, v_1 2) v_1, v_2, v_3 3) v_2, v_4, v_2 4) v_2, v_3, v_1
5) v_3, v_2, v_1 6) v_3, v_1, v_2 7) v_4, v_2, v_3 8) v_4, v_2, v_1

In the above 8 random walks, the number of times a transition from v_2 *to* v_4 *has been made is 1 and the number of times a transition from* v_2 *to* v_1 *(or* v_3*) has been made is 3, whereas, if new random walks are generated from scratch, the two numbers are expected to be the same. The reason for this bias is that in random walks 1, 2, 5, and 6, the walk could not go from* v_2 *to* v_4 *as* v_4 *did not exist when these walks were generated. Note that performing more random walks from each node does not solve the bias problem.*

Take metapath to deal with dynamic embedding



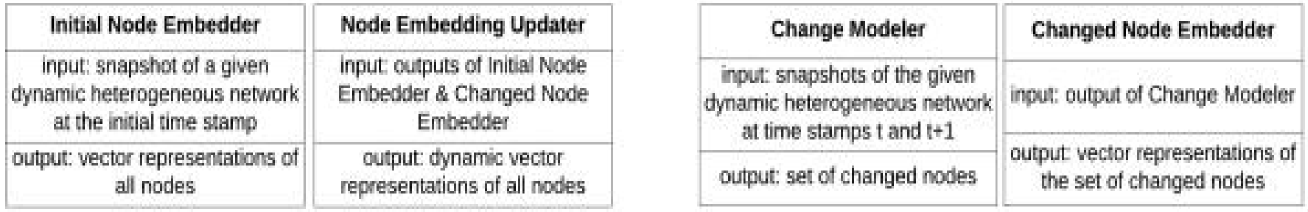
```
Input: The heterogeneous information network $G = (V, E, T)$,
       a meta-path scheme $\mathcal{P}$, #walks per node $w$, walk
       length $l$, embedding dimension $d$, neighborhood size $k$
Output: The latent node embeddings $\mathbf{X} \in \mathbb{R}^{|V| \times d}$
initialize X ;
for $i = 1 \rightarrow w$ do
    for $v \in V$ do
        $MP$ = MetaPathRandomWalk($G$, $\mathcal{P}$, $v$, $l$) ;
        X = HeterogeneousSkipGram(X, $k$, $MP$) ;
    end
end
return X ;

MetaPathRandomWalk($G$, $\mathcal{P}$, $v$, $l$)
$MP[1] = v$ ;
for $i = 1 \rightarrow l{-}1$ do
    draw u according to Eq. 3 ;
    $MP[i+1] = u$ ;
end
return $MP$ ;

HeterogeneousSkipGram(X, $k$, $MP$)
for $i = 1 \rightarrow l$ do
    $v = MP[i]$ ;
    for $j = max(0, i\text{-}k) \rightarrow min(i\text{+}k, l)$ & $j \neq i$ do
        $c_t = MP[j]$ ;
        $X^{new} = X^{old} - \eta \cdot \frac{\partial O(X)}{\partial X}$ (Eq. 7) ;
    end
end
```

ALGORITHM 1: The *metapath2vec++* Algorithm.

⇒



Initial Node Embedder	Node Embedding Updater
input: snapshot of a given dynamic heterogeneous network at the initial time stamp	input: outputs of Initial Node Embedder & Changed Node Embedder
output: vector representations of all nodes	output: dynamic vector representations of all nodes

Change Modeler	Changed Node Embedder
input: snapshots of the given dynamic heterogeneous network at time stamps t and t+1	input: output of Change Modeler
output: set of changed nodes	output: vector representations of the set of changed nodes

Figure 2: Framework of change2vec

1.5 Random Walk Encoders

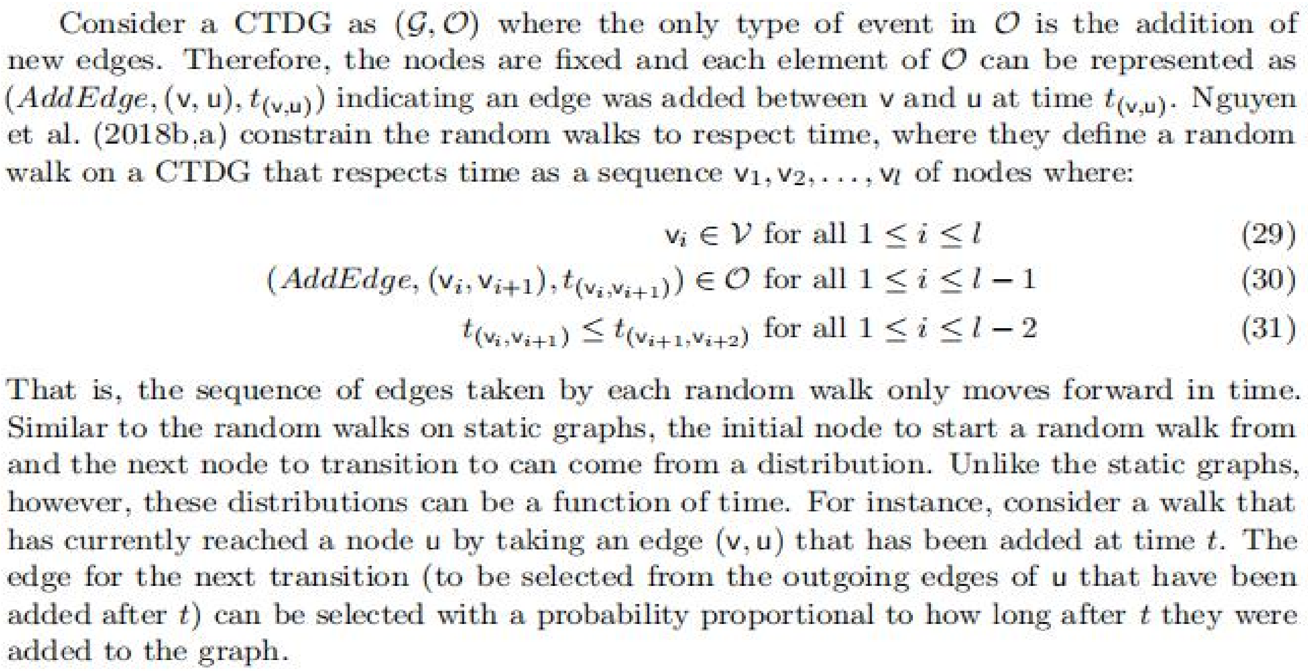
Sajjad(2017) proposed a way to generate unbiased random walks for new snapshots while resusing the previous. (not shown how)

The methods above **may not capture the evolution** and the **temporal patterns** of the nodes.Nguyen(2018) propose an extension of random walk for CTDG that capture temporal patterns of nodes.

Connsider the pair (G, O) where the only type of O is addition of edges., which is represented as (Addedge, (v,u), $t_{(v,u)}$).

They constrain random walk to respect time, they define random walk on CTDG, ehich means the edge selection would be proportional to the time after the edge is added to the graph. see

就是说之前我们在M_{t-1}的基础上找M_{t}是没考虑到M_{t-1}是有一条怎样的路线进化来的，这页slide介绍了考虑M进化的方法。



Consider a CTDG as $(\mathcal{G}, \mathcal{O})$ where the only type of event in $\mathcal{O}$ is the addition of new edges. Therefore, the nodes are fixed and each element of $\mathcal{O}$ can be represented as $(AddEdge, (\mathsf{v}, \mathsf{u}), t_{(\mathsf{v},\mathsf{u})})$ indicating an edge was added between v and u at time $t_{(\mathsf{v},\mathsf{u})}$. Nguyen et al. (2018b,a) constrain the random walks to respect time, where they define a random walk on a CTDG that respects time as a sequence $\mathsf{v}_1, \mathsf{v}_2, \ldots, \mathsf{v}_l$ of nodes where:

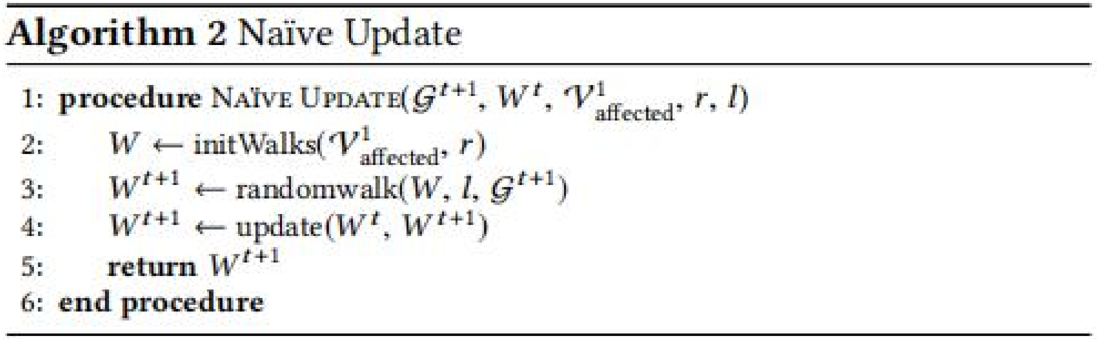
$$\mathsf{v}_i \in \mathcal{V} \text{ for all } 1 \leq i \leq l \tag{29}$$

$$(AddEdge, (\mathsf{v}_i, \mathsf{v}_{i+1}), t_{(\mathsf{v}_i,\mathsf{v}_{i+1})}) \in \mathcal{O} \text{ for all } 1 \leq i \leq l-1 \tag{30}$$

$$t_{(\mathsf{v}_i,\mathsf{v}_{i+1})} \leq t_{(\mathsf{v}_{i+1},\mathsf{v}_{i+2})} \text{ for all } 1 \leq i \leq l-2 \tag{31}$$

That is, the sequence of edges taken by each random walk only moves forward in time. Similar to the random walks on static graphs, the initial node to start a random walk from and the next node to transition to can come from a distribution. Unlike the static graphs, however, these distributions can be a function of time. For instance, consider a walk that has currently reached a node u by taking an edge (v, u) that has been added at time t. The edge for the next transition (to be selected from the outgoing edges of u that have been added after t) can be selected with a probability proportional to how long after t they were added to the graph.

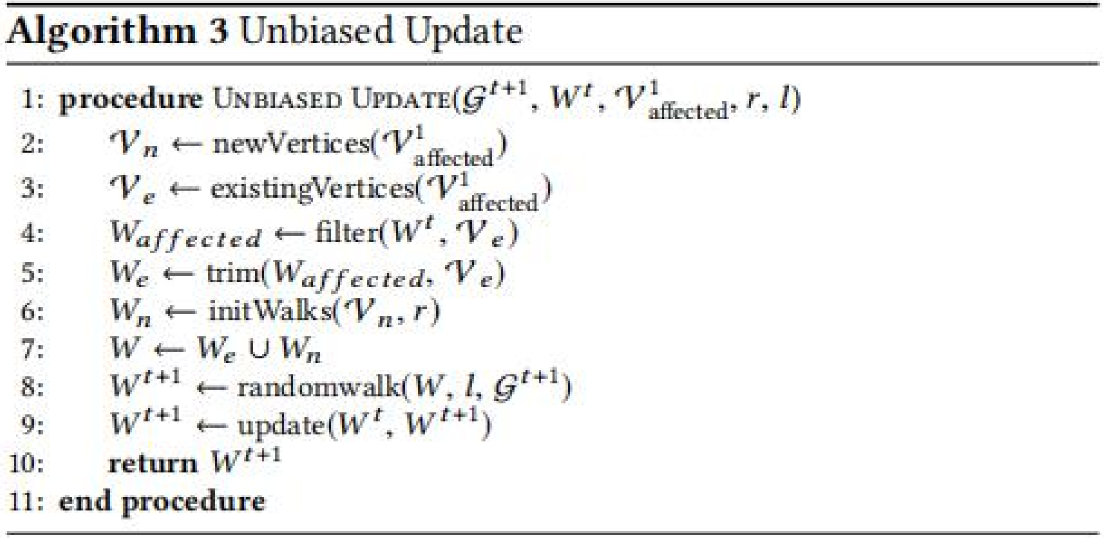
Unbiased random walk?



Algorithm 2 Naïve Update

```
procedure NAÏVE UPDATE($\mathcal{G}^{t+1}$, $W^t$, $\mathcal{V}^1_{\text{affected}}$, $r$, $l$)
    $W \leftarrow$ initWalks($\mathcal{V}^1_{\text{affected}}$, $r$)
    $W^{t+1} \leftarrow$ randomwalk($W$, $l$, $\mathcal{G}^{t+1}$)
    $W^{t+1} \leftarrow$ update($W^t$, $W^{t+1}$)
    return $W^{t+1}$
end procedure
```

replacing the old walks by their corresponding re-generated walks and adding the new random walks for the new vertices



Algorithm 3 Unbiased Update

```
 1: procedure UNBIASED UPDATE($\mathcal{G}^{t+1}$, $W^t$, $\mathcal{V}^1_{\text{affected}}$, $r$, $l$)
 2:     $\mathcal{V}_n \leftarrow$ newVertices($\mathcal{V}^1_{\text{affected}}$)
 3:     $\mathcal{V}_e \leftarrow$ existingVertices($\mathcal{V}^1_{\text{affected}}$)
 4:     $W_{affected} \leftarrow$ filter($W^t$, $\mathcal{V}_e$)
 5:     $W_e \leftarrow$ trim($W_{affected}$, $\mathcal{V}_e$)
 6:     $W_n \leftarrow$ initWalks($\mathcal{V}_n$, $r$)
 7:     $W \leftarrow W_e \cup W_n$
 8:     $W^{t+1} \leftarrow$ randomwalk($W$, $l$, $\mathcal{G}^{t+1}$)
 9:     $W^{t+1} \leftarrow$ update($W^t$, $W^{t+1}$)
10:     return $W^{t+1}$
11: end procedure
```

the Unbiased Update algorithm finds the affected walks (Line 4) and trims them to the first affected vertex (Line 5). After that, Unbiased Update resumes the trimmed random walks until they are of the given length l.

1.5 Analysis of Random Walk Encoder

Supervised and unsupervised learning:

Similar to decomp. and autoencoder methods, the main advantage is that they **provide embedding** function **without needing to be combined with a decoder**. Hence, it could be used in unsupervised learning such as **clustering and community detection.**
Disconnectness between encoder and decoder prevents the model from being trained end to end. So it's not best in supervised learning.

Streaming scenario:

When new observation made, random walk require to take new walks in account and update embedding. This update could be **time consuming**, which makes random walk not idealistic for stream scenario.

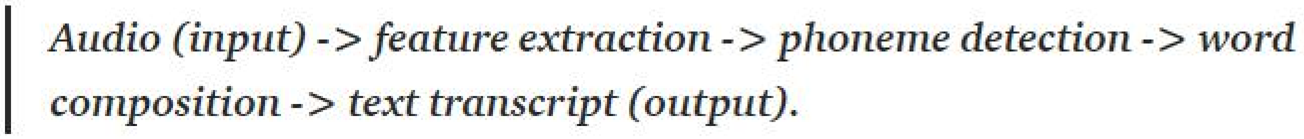
Random walk for attributed graph:

Using random walks for representation learning has been mostly done for non-attributed graphs. Random walk for **attributed graph** would be a interesting future work.

End to end training?

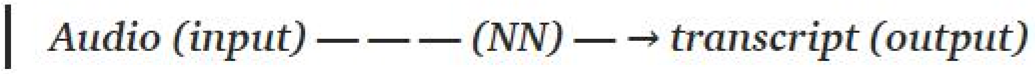
End-to-end (E2E) learning refers to training a possibly complex learning system represented by a single model (specifically a Deep Neural Network) that represents the complete target system, bypassing the intermediate layers usually present in traditional pipeline designs.

音频转文本



> *Audio (input) -> feature extraction -> phoneme detection -> word composition -> text transcript (output).*

↓



> *Audio (input) ——— (NN) —→ transcript (output)*

1.6 Sequence Model Encoder

What?

- **TOPIC**:Deal with **asynchronous sequence modling problem** based on **RNN.**
- **TASK**: mainly for **extra**polation.
- **REASON**: go through observations sequentially and provide embedding at current time based on the past.
- **EXCEPTION: Inter** task examples.

bi-directional sequence model one runing forward and provide embedding based on everything before time t and the other running backward and providing embedding everything after t.

GCN/RNN 基本结构

GCN：对节点embedding用的(GCN分为spectralGCN和spatialGCN两种。下面这公式是spectral GCN的一种)

$$\begin{aligned} H_t^{(l+1)} &= \mathrm{GCONV}(A_t, H_t^{(l)}, W_t^{(l)}) \\ &:= \sigma(\widehat{A}_t H_t^{(l)} W_t^{(l)}), \end{aligned}$$

RNN：用来捕捉节点演化的时序信息的。（主要是LSTM 和GRU两种常用）

一般用GCN对节点进行embedding，然后将这些embedding信息作为RNN的输入。

1.6.1 RNN-Based Encoders For DTDG

Consider $\{G^1 G^2 \ldots G^T\}$ and M be a encoder, which given G^t, the representation for each node is out. For example, M could be GCN.

One way for using RNN for DTDG: run M on each G^t and we get sequence

$$z_v^1, z_v^2, \ldots, z_v^T$$

The sequence is then fed in decoder to make predictions. It is like static feature aggregation except that the weights of RNN and model M are learned **simultanuously** and over all snapshots.

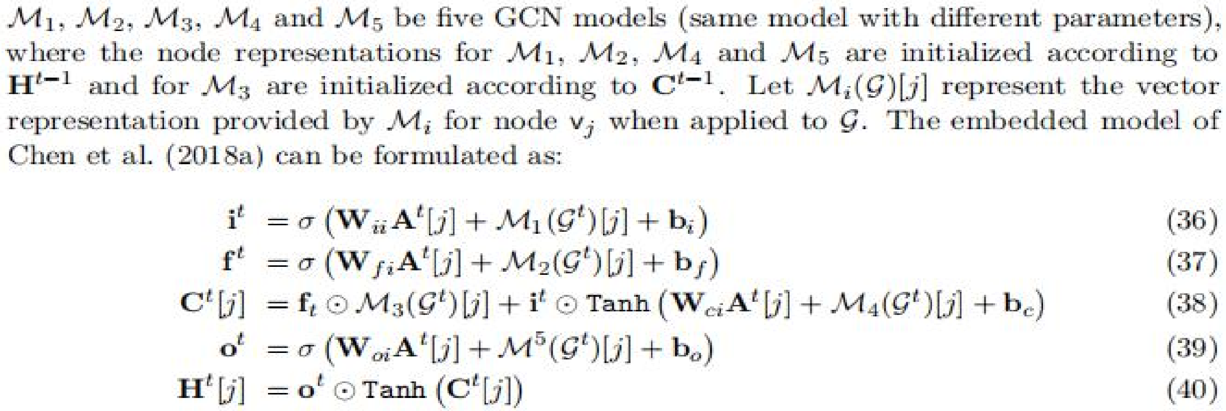
The t_th step of the encoder is

$$z_{v_1}^t z_{v_2}^t, \ldots, z_{v_{|V^t|}}^t = M(G^t)$$
$$h_{v_j}^t = RNN(h_{v_j}^{t-1}, z_{v_j}^t) for\ j \in [1, |V^t|]$$

where h represents hidden state of RNN corresponding to vec representation of size d for $|V^t|$ nodes in graph that capture history of nodes as well.
M aims at capturing structural info.

1.6.1 RNN-Based Encoders For DTDG

Some models based on it:
...Some who used GCN as M and LSTM as RNN.
Yu (2019) used 3D GCN that aggregates **features of neighboring node**s on a window of previous snapshots.(i.e. the aggregaio is bothh spatial and temporal than just spatial)
In above M is independent of RNN which means vector representation for nodes provided by M are independent of node histories captured by $h_{v_j}^{t}$.
The embeded approaches aim at **embedding model M into RNN so that M can also use node histories**.
One such embedded approach has been proposed by Chen et al. (2018a), where the authors combine the GCN proposed by Defferrard et al. (2016) with LSTMs. Consider a DTDG as {G1, G2, . . . , GT }, let At represent the adjacency matrix for Gt, and let At[j] represent the jth row of At. Let $C^{t-1}[j]$ and $H^{t-1}[j]$ represent the **memory and hidden state of the LSTM** at time t – 1 for node vj.



$\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, $\mathcal{M}_4$ and $\mathcal{M}_5$ be five GCN models (same model with different parameters), where the node representations for $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_4$ and $\mathcal{M}_5$ are initialized according to $\mathbf{H}^{t-1}$ and for $\mathcal{M}_3$ are initialized according to $\mathbf{C}^{t-1}$. Let $\mathcal{M}_i(\mathcal{G})[j]$ represent the vector representation provided by $\mathcal{M}_i$ for node v_j when applied to $\mathcal{G}$. The embedded model of Chen et al. (2018a) can be formulated as:

$$\mathbf{i}^t = \sigma\left(\mathbf{W}_{ii}\mathbf{A}^t[j] + \mathcal{M}_1(\mathcal{G}^t)[j] + \mathbf{b}_i\right) \tag{36}$$

$$\mathbf{f}^t = \sigma\left(\mathbf{W}_{fi}\mathbf{A}^t[j] + \mathcal{M}_2(\mathcal{G}^t)[j] + \mathbf{b}_f\right) \tag{37}$$

$$\mathbf{C}^t[j] = \mathbf{f}_t \odot \mathcal{M}_3(\mathcal{G}^t)[j] + \mathbf{i}^t \odot \mathrm{Tanh}\left(\mathbf{W}_{ci}\mathbf{A}^t[j] + \mathcal{M}_4(\mathcal{G}^t)[j] + \mathbf{b}_c\right) \tag{38}$$

$$\mathbf{o}^t = \sigma\left(\mathbf{W}_{oi}\mathbf{A}^t[j] + \mathcal{M}^5(\mathcal{G}^t)[j] + \mathbf{b}_o\right) \tag{39}$$

$$\mathbf{H}^t[j] = \mathbf{o}^t \odot \mathrm{Tanh}\left(\mathbf{C}^t[j]\right) \tag{40}$$

略

1.6.1 RNN-based Encoders for DTDG

Other similar ways :

Pareja(2019) embed RNN into GCN by running GCN with different parameters at each snapshot where RNN provide weights of GCN at t th snapshot based on weights of GCN at previous snapshots.

1.6.2 Other Sequence Model Encoders for DTDG

Sarker(2007) used **kalman filter** to track embedding of the nodes through time for bipartite graph.Each timestep of Kalman filter corresponds to a snapshot of DTDG.

The t th timestamp observes the adjacency matrix A^t and update node embedding accordingly. To make computation tractable, the conditional probability of observation (i.e. Adjacency matrix) given the hidden state is approximated by a Gaussian.

Inspired by fully attentative model for DTDG, consider standard DTDG, DySAT applied attention-based GCN of Velickovic (2018) to each G^t and obtain $z_v^1 z_v^2, \ldots, z_v^t$ for every node.DySAT added a positional embedding p^t (which encodes informationn abouut relation position of tth snapshot compared with other snapshots)to embedding z_v^t and get $\bar{z}_v^t = z_v^t \ + p^t$.

Finally, DySAT applied multi-head self-attention on $\bar{z}_v^1 \bar{z}_v^2, \ldots, \bar{z}_v^T$ to get the final representation of mode to be sent to decoder.
Kazemi(2019) extend positional encoding to continuous time encoding through vector representation for time dubbed Time2Vec. Interesting direction: **extend DySAT to CTDG by replacing positional encodinng with Time2Vec**

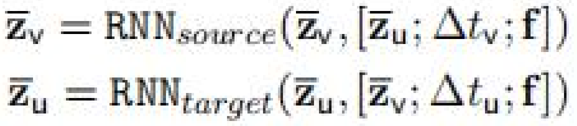
略

1.6.3 RNN-Based Encoders For CTDGs

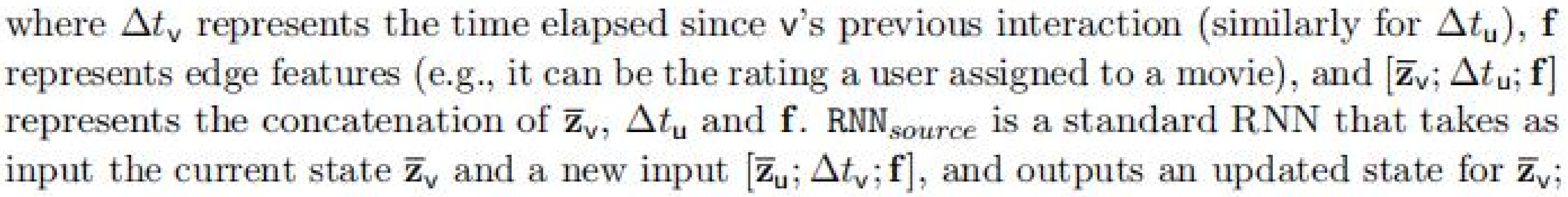
Restraint： Only consider adding new edges.
Difference: the way they define embedding function and custom RNN.
Example;

Kumer(2018) developed JODIE, which define EMB(v) = $(z_v, \bar{z}_v)$ for each node v, and $z_v \in \mathbb{R}^{d1}, \bar{z_v} \in \mathbb{R}^{d2}$. The value of z_v is directly optimized as shallow encoder and but $\bar{z}_v$ is updated by **RNN**.

In JODIE, use 2 different RNN to update source and target node. Upon new obserbvation (AddEdge, (v, u), t), then



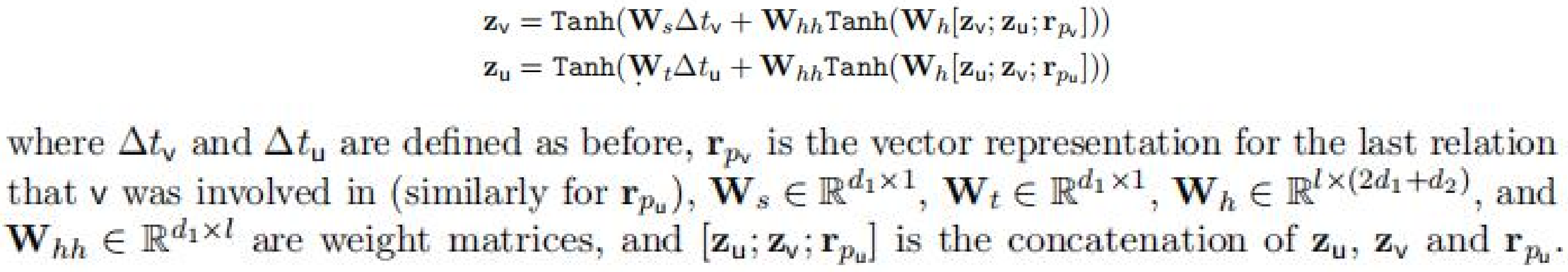
$$\bar{\mathbf{z}}_{\mathsf{v}} = \mathrm{RNN}_{source}(\bar{\mathbf{z}}_{\mathsf{v}}, [\bar{\mathbf{z}}_{\mathsf{u}}; \Delta t_{\mathsf{v}}; \mathbf{f}])$$
$$\bar{\mathbf{z}}_{\mathsf{u}} = \mathrm{RNN}_{target}(\bar{\mathbf{z}}_{\mathsf{u}}, [\bar{\mathbf{z}}_{\mathsf{v}}; \Delta t_{\mathsf{u}}; \mathbf{f}])$$



where Δt_{v} represents the time elapsed since v's previous interaction (similarly for Δt_{u}), $\mathbf{f}$ represents edge features (e.g., it can be the rating a user assigned to a movie), and $[\bar{\mathbf{z}}_{\mathsf{v}}; \Delta t_{\mathsf{u}}; \mathbf{f}]$ represents the concatenation of $\bar{\mathbf{z}}_{\mathsf{v}}$, Δt_{u} and $\mathbf{f}$. RNN_{source} is a standard RNN that takes as input the current state $\bar{\mathbf{z}}_{\mathsf{v}}$ and a new input $[\bar{\mathbf{z}}_{\mathsf{u}}; \Delta t_{\mathsf{v}}; \mathbf{f}]$, and outputs an updated state for $\bar{\mathbf{z}}_{\mathsf{v}}$;

1.6.3 RNN-Based Encoders For CTDGs

Trevedi(2017) consider **KG** and define EMB(v) = (z_v), EMB(r) = (z_r) where they may not in same dimesion. Their model **KnowEvolve**, define two custom RNN to update z_v,z_u upon observation (AddEdge, (v, r, u), t).



$$\mathbf{z}_{\mathsf{v}} = \mathrm{Tanh}(\mathbf{W}_s \Delta t_{\mathsf{v}} + \mathbf{W}_{hh}\mathrm{Tanh}(\mathbf{W}_h[\mathbf{z}_{\mathsf{v}}; \mathbf{z}_{\mathsf{u}}; \mathbf{r}_{p_{\mathsf{v}}}]))$$
$$\mathbf{z}_{\mathsf{u}} = \mathrm{Tanh}(\mathbf{W}_t \Delta t_{\mathsf{u}} + \mathbf{W}_{hh}\mathrm{Tanh}(\mathbf{W}_h[\mathbf{z}_{\mathsf{u}}; \mathbf{z}_{\mathsf{v}}; \mathbf{r}_{p_{\mathsf{u}}}]))$$

where Δt_{v} and Δt_{u} are defined as before, $\mathbf{r}_{p_{\mathsf{v}}}$ is the vector representation for the last relation that v was involved in (similarly for $\mathbf{r}_{p_{\mathsf{u}}}$), $\mathbf{W}_s \in \mathbb{R}^{d_1 \times 1}$, $\mathbf{W}_t \in \mathbb{R}^{d_1 \times 1}$, $\mathbf{W}_h \in \mathbb{R}^{l \times (2d_1 + d_2)}$, and $\mathbf{W}_{hh} \in \mathbb{R}^{d_1 \times l}$ are weight matrices, and $[\mathbf{z}_{\mathsf{u}}; \mathbf{z}_{\mathsf{v}}; \mathbf{r}_{p_{\mathsf{u}}}]$ is the concatenation of $\mathbf{z}_{\mathsf{u}}$, $\mathbf{z}_{\mathsf{v}}$ and $\mathbf{r}_{p_{\mathsf{u}}}$.

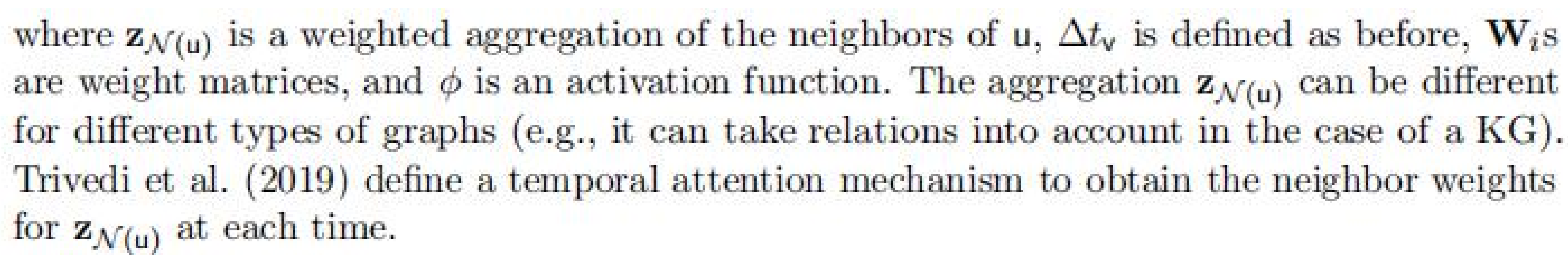
Vector representation for relations is **optimized directly**.
Compared to JODIE, KnowEvolve may be more influenced by Δt because **KnowEvolve** projects each Δt to a vector and sums the resulting vector with influence coming from **representation of source, target and relation**. Unlike JODIE, dependence of the update rule on Δt in Know-Evolve is somewhat ***separate from* the two nodes and the relation involved.**

（**略**）Trevedi(2019) developed model that can be used in **several types of graphs**. They define EMB(v) = (z_v). Upon new observation (AddEdge, (v, u), t), update node representation for v using

$$z_v = \varphi(W_1 z_{N(u)} + W_2 z_v + W_3 \Delta t_v)$$

1.6.3 RNN-Based Encoders For CTDGs

略



where $\mathbf{z}_{\mathcal{N}(\mathsf{u})}$ is a weighted aggregation of the neighbors of u, Δt_{v} is defined as before, $\mathbf{W}_i$s are weight matrices, and ϕ is an activation function. The aggregation $\mathbf{z}_{\mathcal{N}(\mathsf{u})}$ can be different for different types of graphs (e.g., it can take relations into account in the case of a KG). Trivedi et al. (2019) define a temporal attention mechanism to obtain the neighbor weights for $\mathbf{z}_{\mathcal{N}(\mathsf{u})}$ at each time.

1.6.4 Discussion of RNN-based Encoders

Information propagation:existing work almost just update the end node of newly added edges but Ma(2018) argue that the propagation should be considered, at least neighborhoods.
They compute vector representation for new observation as below:

$$z_o = \varphi(W_1 z_{s_v} + W_2 z_{t_u} + b)$$

where z_{s_v}, z_{t_u} belong to EMB(v) and EMB(u), $W_1 W_2$ and b are learnable parameters. z_o is sent to immediate neighbors of v and u and custom RNNs update the repersentation f neighbors based on z_o and how they are connected to v or u.

Attributed graph: Now the nodes have attributes with fixed values, one way to incorporate these attributes is by initializing node representations using attributed values.(Trevidi 2019)
For graph that attribute could change, Seo(2018) made models to take those changes in account for DTDG. It is a **interesting future direction**.

Streaming Scenario:RNN-based approachs for CTDG, once RNN weights are learned during training, RNN has learned how to take observation as input and update node embedding **without requiring computing gradients.**
This makes RNN-based approchs a **natrual choice for streaming scenario.**
Although as data collected during test increasing (when reach some predefined **threshold**), training can run again on all collected data to **learn better weights for RNN**, then the weight can be **frozed again** and updated RNN can replace old one.

1.7 Autoencoder based Encoders

Consider standard DTDG, let A^t be the adjacency matrix. Goyal(2017) learn autoencoder AE^1 for G^1 similar to SDNE, where encoder takes input $A^1[i]$ and generate vector representation z_{vi}^1 for node v_i . Reconstructer takes z_{vi}^1 as input and reconstructs $A^1[i]$. $z_{vi}^1 z_{vj}^1$ are constrained to be near if linked.Having AE^1 ,for the tth snapshot , an **autoencoder AE^t is based on the parameters of AE^{t-1} and trained based on A^t** , to produce vector representation.AE^t can have different structure to AE^{t-1} (like neurons/layers), the size is decided by heuristic methods according to AE^{t-1} and how different between G^t&G^{t-1}. If size are different , then initialie with AE^{t-1} by method of Net2WiderNet and NetDeeperNet approaches (Chen(2015)).

Goyal’s method
1. use info. within previous snapshots of DTDG to enable learning autoencoder for current snapshot faster.
2. AE^{t-1} implicitly acts as a regularizer imposing smooth constraint.
3. may not capture evolution of nodes.

1.7 Autoencoder based Encoders(略)

To better capture evolution, Bonner(2018) propose to develop autoencoder that reconstruct node's neighborhood in next snapshot given the previous one.

They also propose variational sutoencoder model :instead of directly learning embedding Z^t in the encoder, they learn a Gaussian distribution from which Z^t is sampled. The Gaussian parametrized by $\mu\&\gamma$, which are laerned by seperated two two-layer GCN with tied parameters on first layer.

To take more snapshots into account in learning node embedding Goyal(2018) propose to learn a single autoencoder where at snapshot t, encoder takes input

$$A^{t-l}[i], A^{t-l+1}[i], \dots, A^{t-1}[i]$$

and produce vector $z_{v_i}^t$, reconstructer takes it as input to reconstruct $A^t[i]$

Example for encoder:
1. feeding concatenation $[A^{t-l}[i]; A^{t-l+1}[i]; \dots,, A^{t-1}[i]]$ into a feedforward neuronetwork
2. feed sequence $A^{t-l}[i]; A^{t-l+1}[i]; \dots,, A^{t-1}[i]$ to LSTM.

Similar architecture is used for reconstructer.

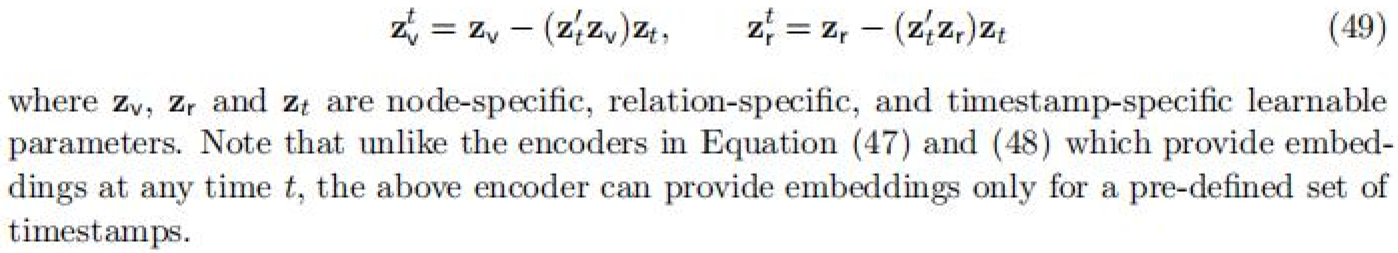
1.8 Diachronic Encoders（略）

Definition: encoders that map every pair (node, timestamp) or (relation, timestamp) to a hidden representation. Such encoers can be used effectively for interpolation as they learn to provide node and relation embeding **at any point of time**. Goel(2020) propose a diachronic encoder for nodes of a KG in which

$$\mathbf{z}_{\mathsf{v}}^{t}[i] = \begin{cases} \mathbf{a}_{\mathsf{v}}[i]\phi(\mathbf{w}_{\mathsf{v}}[i]t + \mathbf{b}_{\mathsf{v}}[i]), & \text{if } 1 \le i \le \gamma d. \\ \mathbf{a}_{\mathsf{v}}[i], & \text{if } \gamma d < i \le d. \end{cases}$$

where $a_v \in \mathbb{R}^d$, $w_v \in \mathbb{R}^{\gamma d}$ and $b_v \in \mathbb{R}^{\gamma d}$ are node--specific parameter, γ is percentage of feature that are a function of time.

Xu (2019) defined



$$\mathbf{z}_{\mathsf{v}}^{t} = \mathbf{z}_{\mathsf{v}} - (\mathbf{z}_t'\mathbf{z}_{\mathsf{v}})\mathbf{z}_t, \qquad \mathbf{z}_{\mathsf{r}}^{t} = \mathbf{z}_{\mathsf{r}} - (\mathbf{z}_t'\mathbf{z}_{\mathsf{r}})\mathbf{z}_t \tag{49}$$

where $\mathbf{z}_{\mathsf{v}}$, $\mathbf{z}_{\mathsf{r}}$ and $\mathbf{z}_t$ are node-specific, relation-specific, and timestamp-specific learnable parameters. Note that unlike the encoders in Equation (47) and (48) which provide embeddings at any time t, the above encoder can provide embeddings only for a pre-defined set of timestamps.

1.8 Diachronic Encoders（略）

Instead of consider shallow embeddinng for nodes, Dasgupta(2018) consider **shallow embeddinng for characters** in the timestamps. Then they map (relation, timestamp) into vector representation of z_r^t by

$$\mathbf{z}_\mathrm{r}^t = \mathrm{LSTM}(\mathbf{z}_\mathrm{r}, \mathbf{z}_{c_1}, \ldots, \mathbf{z}_{c_k})$$

c_i is the ith character in t. It can natrually deal with missing value in dates. It can also used for node embedding.

1.9 Staleness（略）

Consider extra. problem over CTDG and an encoder that updates embedding for node v whenever a new observation involving v is made. Assume the last time the encoder updated the embedding for v was t_v and now it is t. Depending on how long the time passed, the embedding of v may be staled.

To handle staleness of repre, Kumer(2017) propose mathod to learn how repre of v involves when no observation involving v is made. He first create a vector repre $z_{\Delta t_v}$ for Δt_v

$$\mathbf{z}_{\Delta t_v}[i] = \mathbf{w}[i]\Delta t_v + \mathbf{b}[i]$$

w and b are vectors withlearnablee para. Then compute

$$\mathbf{z}_v^t = (1 + \mathbf{z}_{\Delta t_v}) \odot \mathbf{z}_v$$

Instead use z_v which may be staled, they use z_v^t as embedding. Althogh diachronic encoder usually used for inter, but when used for extra, there is **benefit of updating node and relation embedding** even no observation have been made about node and relation.

中国科学院大学
University of Chinese Academy of Sciences

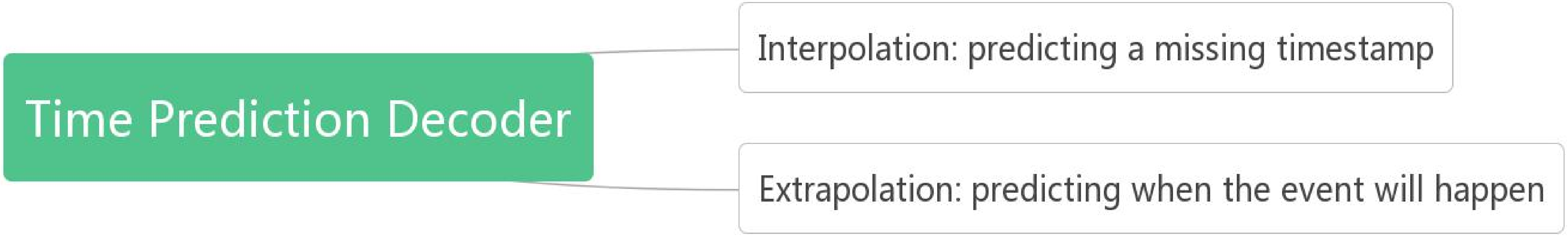
/02

2. Decoder of Dynamic Graph

2.1 Time-Predicting Decoder

2.2 Time-Conditioned Decoder

2.1 Time-Predicting Decoders



Time Prediction Decoder

Interpolation: predicting a missing timestamp

Extrapolation: predicting when the event will happen

2.1 Time-Predicting Decoders

Temporal Point Process

a **stochastic process** used for modeling **sequential asynchronous discrete events** occuring in continuous time.

For a sequence of discrete event happened at $t_1, \ldots, t_n$, $t_n \leq T$,TPP is characterised by using a **conditional intensity** function $\lambda(t)$ such that $\lambda(t)dt$ represents probability of an event happening in this interval [t, t+dt].given $t_1, \ldots, t_n$ and no event happen during $t_n < t \leq T$.

Then **conditional density** of occurance of **next event** at some time point in $t_n < t \leq T$ is $f(t) = \lambda(t)S(t)$,where $S(t) = exp(-\int_{t_n}^{t} \lambda(\tau)d\tau)$ is called **survival function**, means the prob. of no events happenning during $[t_n, t)$. Time of next event can be estimated as the expectation over f(t).

Design of intensity function is hand designed, except (Du et al) use totally data to learn intensity function.

TPP with intensity function parametrised by the node representation in graph can be constructed to **predict when** something would **happen** to a single node.

2.1 Time-Predicting Decoders

First, find all paths between two nodes. Paths are matched with predefined path templates（模板） and paths matching each template is counted. These counts are considered as node-pair embedding.

Then, feed the embedding into generalised linear model and the **score of the model** is used to **define parameter** (like the θ below) of a density function.

The distribution for formation of a relation between nodes is decided by various distributions(**human** decided) like

$$f(t) = \frac{1}{\theta} exp(-\frac{t}{\theta})$$

Expectation of t~f(t) can be used to **predict when relation will be formed** between two nodes.

2.1 Time-Predicting Decoders

Some recent variations:

Trivedi et al (2017) consider an encoder provides embedding such that given a dynamic graph until t gives EMB(v) = (z_v^t) and EMB(r) = (P_r), define the relation formation

$$S_{v,r,u}(t) \; = \; z_v^{t'} P_r z_u^t$$

给了一种lambda的构造方式

then intensity function could be formed as

$$\lambda_{v,r,u}(t|H_{t-}) \; = \; exp(S_{v,r,u}(t))(t \; - \; \bar{t})$$

here H_{t-} is the history until t without t included.$\bar{t}$ is the most recent time that either v or u is involved in observation(t > $\bar{t}$), convert $\lambda_{v,r,u}$ to conditional density function ($f_{v,r,u}$) and then take expectation.

p.s. It is not allowed here for concurrent events, if allowed, it would contradict when $t = \bar{t}$.

2.1 Time-Predicting Decoders

When **different types of relations evolving** at different rates is considered, (e.g. liking posts in social network would evolve much faster than becoming friends).Model the dynamics considering 2 types of realtions:

1. communications corres. to node interactions(liking posts)
2. association correds to topological evolutions(becoming friends)

They use the embedding EMB(v) = (z_v^t) and EMB(r) = (ψ_r, z_r^t) and define intensity function as

$$\lambda_{v,r,u}(t|H_{t-}) = \psi_r log(1 + exp(\frac{z_r^{t'}[z_v^t; z_u^t]}{\Psi_r}))$$

where Ψ_r is **evolution rate** decided by different types introduced.

2.1 Time-Predicting Decoders

Zuo et al(2018) use intensity function of a Hawkes process(1971) to do predictions.

For interpolation in KGs, to predict a timestamp for triple (v, r, u, ?),Leblay and Chekol(2018) replace the missing timestamp with all timastamps observed in this KG and then find scores for all produced triples using **time conditioned decoder**s. Note this may not be used in KG with **many timestamps**.

Q: Why can not be used for large amount of timestamps?

略

2.1 Time-Conditioned Decoders

These decoders aim at **making predictions for specific timestamps** given as input. These can be used for

1. extrapolation (e.g. predict who will be CEO of Apple two y from now)
2. interpolation(e.g. predict who was the CEO of Apple on 2006-04-01, assuming this info is missing in the known KG)

Dasgupta et al.(2018) developed model for intra. in KG. They use encoder by

$$z_v^t = z_v - (z'_t z_v) z_t$$

$$z_r^t = z_r - (z'_t z_r) z_t$$

To predict whether there exists some edge (v, r, u) at time t or not, use “TransE” as decoder:

$$||z_v^t + z_r^t - z_u^t||$$

or define decoder as

$$||z_v + z_r + z_t - z_u||$$

2.1 Time-Conditioned Decoders

If shallow encoder's used for timestamp embeddings , then an embedding can **only be learned** for timestamps that **have been observed** in train set.

This approach **may not generalize well** to timestamps not observed in train set as vector representation has not been learned for these timestamps.

For example, If KG does not contain any info with timestamp 08/07/2012, then this approach **do not learn an embedding for this date** and may not do any prediction about that date.

For this reason, this can not be used for extra.

Question: Why the embedding of time could not be learned when no observation occured?

2.1 Time-Conditioned Decoders

To address the problem, Garefa-Duran(2018) use encoder

$$z_r^t = LSTM(z_r, z_{c_1}, \ldots, z_{c_k})$$

where c_i is the ith character in timestamp t.

Having z_v, z_r^t, z_u, use TransE and DistMult as decoder. **Since they use shallow embedding for each character in the timestamp (not for timestamp itself), the model can be potentially applied to timestamps unseen during training.**

2.1 Time-Conditioned Decoders

When consider more, we find that just **using** static node representation $\boldsymbol{z_v}$ **is not enough**. Becasuse to make prediction of a node v's behavior at time t, (e.g. predicting the movies v like in the year of 1990), one need to know **specific properties** of v around that time but **not an aggregation** along the whole time period.

To address, Goel(2020) use encoder as

$$z_v^t[i] \; = \begin{cases} a_v[i]\emptyset(w_v[i]t + b_v[i]) \\ \qquad\qquad a_v[i] \end{cases}$$

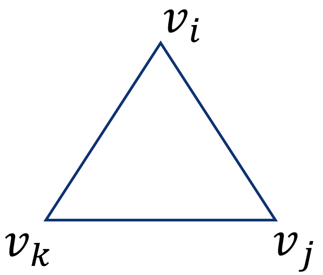
where $a_v \in R^d$,$w_v, b_v \in R^{\gamma d}$ are (entity specific) vectors with learnable parameters. Question: Where do I get $a_v \in R^d$,$w_v, b_v \in R^{\gamma d}$? What are them specifically?

They proved using sine as ∅ and SimplE as decoder results in **fully expressive model for link prediction** in temporal KG.

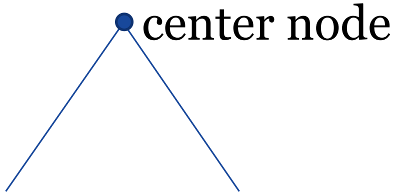
2.1 Time-Conditioned Decoders

Making Predictions for A Single Timestamp

In case where all predictions are to be made for a single timestamp or a single time interval,(e. predicting what happens in the next snapshot of DTDG, or predicting what hapens in the near future without predicting when it will happen)
Zhou(2018) made link prediction employing a triadic closure based point process for link prediction. Let v_i, v_j, v_k be 3 nodes in a graph at t, they form a **closed triad** if all of them are pair-wise connected, **open triad** if all but one pair is connected.



closed triad



open triad

Fundimental machenism in formation and evolution of dynamic networks is **triad closure.** it is the preocess of closed triads being created from open triads.

The probability of linking v and u is proportional to

1. the number of tirads the link would close
2. the similarity of z_u^t ,z_v^t to the embedding of center node

中国科学院大学
University of Chinese Academy of Sciences

Thanks for Listening

Yinhan He
UCAS