CoPE: Modeling Continuous Propagation and Evolution on Interaction Graph

Yao Zhang*
yaozhang@fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
Shanghai, China

Yun Xiong†
yunx@fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
Shanghai Institute for Advanced Communication and Data Science
Shanghai, China

Dongsheng Li
Caihua Shan
Kan Ren
dongsli@microsoft.com
caihuashan@microsoft.com
kanren@microsoft.com
Microsoft Research Asia
Shanghai, China

Yangyong Zhu
yyzhu@fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
Shanghai Institute for Advanced Communication and Data Science
Shanghai, China

ABSTRACT

Human interactions with items are being constantly logged, which enables advanced representation learning and facilitates various tasks. Instead of generating static embeddings at the end of training, several temporal embedding methods were recently proposed to learn user and item embeddings as functions of time, where each entity has a trajectory of embedding vectors aiming to encode the full dynamics. However, these methods may not be optimal to encode the dynamical behaviors on the interaction graphs in that they can not generate "fully"-temporal embeddings and do not consider information propagation. In this paper, we tackle the issues and propose CoPE (**Co**ntinuous **P**ropagation and **E**volution). We use an ordinary differential equation based graph neural network to model information propagation and more sophisticated evolution patterns. We train CoPE on sequences of interactions with the help of meta-learning to ensure fast adaptation to the most recent interactions. We evaluate CoPE on three tasks and prove its effectiveness.

CCS CONCEPTS

• **Computing methodologies** → **Learning latent representations**; *Neural networks*; • **Information systems** → Recommender systems.

KEYWORDS

interaction graph, temporal embedding, graph neural networks

ACM Reference Format:
Yao Zhang, Yun Xiong, Dongsheng Li, Caihua Shan, Kan Ren, and Yangyong Zhu. 2021. CoPE: Modeling Continuous Propagation and Evolution on Interaction Graph. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482419

1 INTRODUCTION

Human interactions with other objects are constantly being logged and collected. These interactions comprise rich information which could facilitate a series of related tasks like anomalous activity detection and future interaction prediction. Within these applications, representation learning [20, 26, 27] lays the foundation of many recent state-of-the-art algorithms. Some methods [18] learn low-dimensional vector representations, *i.e.*, embeddings, for users and items by matrix factorization. Considering that user-item interactions can be organized as a timestamped bipartite graph called interaction graph (Figure 1(a)), some methods utilize graph neural networks (GNNs) [11, 22] to learn more powerful representations by capturing information from connectivities. Once we have obtained the representations, probabilities of interacting can be computed from the corresponding user and item representations with simple functions, *e.g.*, multi-layer perceptrons or inner product.

These methods assume user interests and item features are constant and only assign static embeddings to users and items at the end of training. This is contrary to the dynamic nature of the real world, where interactions happen sequentially. As shown in the literature [5, 9, 19], users and items are evolving over time. For instance, the drifts of user interests can be reflected by items he/she bought. Item features, *e.g.*, popularity and seasonal variations, are also changing over time. Therefore, evolution methods including

*This work was done when the author was an intern with Microsoft Research Asia.
†Corresponding author

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482419

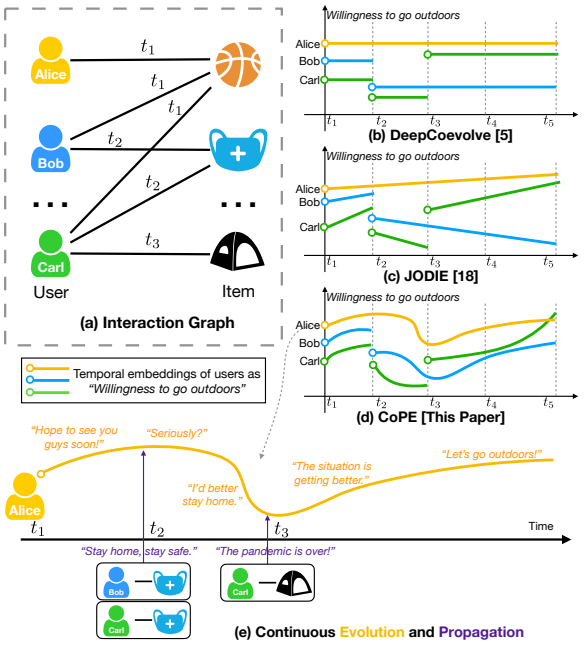


Figure 1: An example interaction graph (a) and temporal embeddings of user nodes generated by different methods (b-d). Circles on the curves represent discontinuity points where interactions have direct impacts on the nodes. For illustration purposes, the embedding is viewed as the *willingness to go outdoors*. (a) An interaction graph is a user-item bipartite graph with timestamped edges. (b) DeepCoevolve [5] generates piecewise constant embeddings. (c) JODIE [19] uses linear functions to estimate embeddings between observations. These two models would not update embeddings until relevant interactions occur. For example, Alice was still willing to go outdoors even a pandemic outbroke at t_2 since there is no other interactions involving Alice. (d) We propose CoPE utilizing an ordinary differential equation based graph neural network to model information propagation and more sophisticated evolution patterns. (e) We annotate Alice's embedding in Figure 1(d) to illustrate the continuous evolution and propagation on the interaction graph.

DeepCoevolve [5] and JODIE [19] are proposed to learn temporal embeddings of users and items, *i.e.*, embeddings that are functions of time. As such, users and items could have trajectories of embedding evolving over time, which encodes their full dynamics.

However, most existing methods fail to learn "fully" temporal embeddings and do not consider information propagation. They assume that an interaction would only influence the involved user and item. For instance, DeepCoevolve [5] would not update a user's (or an item's) embedding until a relevant interaction occurs, resulting in piecewise constant embeddings as shown in Figure 1(b); though JODIE introduces a linear projection operation (Figure 1(c)) trying to predict future embeddings, it has the same issue. The embeddings produced by DeepCoevolve or JODIE follow the same trend after the last relevant interaction even when the world may have changed dramatically. For example, we may explain one dimension of embeddings as the *willingness to go outdoors* as shown in Figure 1(b)-(c). Since there is no interaction after t_1 for Alice, she still wanted to go outdoors even a pandemic outbroke at t_2. Similarly, Bob would not know the pandemic was over at t_3 and kept the low willingness. For both DeepCoevolve and JODIE, a node only involved in an earlier interaction would not be updated anymore and can hardly be aware of how the world is going.

As pointed out by [13], people are influenced by others when making decisions. Even without social networks, these connections among users can be inferred from the interaction graph, on which impacts of interactions would be spread. So an interaction would eventually influence other users and items, *i.e.*, information propagation. On the other hand, the continuous evolution patterns are more than linear in the real world. Take the embedding (*i.e.*, willingness to go outdoors) of Alice as an example as illustrated in Figure 1(e). After playing basketball at t_1, Alice's willingness was growing steadily. But at t_2 an pandemic outbroke, and Bob and Carl bought masks and told Alice this information. Alice might underestimate it but then realized the severity. At t_3 the pandemic was over, and Carl went camping. This information was propagated to Alice, and Alice recovered her willingness. DeepCoevolve and JODIE fail to capture these phenomena.

In this paper, we tackle the above issues and propose CoPE (**Co**ntinuous **P**ropagation and **E**volution). As the name implies, we model both continuous propagation and evolution simultaneously on the interaction graph as illustrated in Figure 1(d) and (e). Specifically, the most important component of CoPE is the continuous propagation and evolution unit. It utilizes an ordinary differential equation (ODE) [4] based graph neural network, CGNN [33], to evolve node representations and propagate information continuously. Since the continuous nature is modeled by an ODE, we can model more sophisticated patterns of evolution than linear ones in JOIDE. Meanwhile, the propagation is also modeled thanks to the graph neural network. As such, interactions like (Bob, Masks) and (Carl, Masks) would gradually influence Alice.

Another important component is the discrete update unit modeling immediate impacts of interactions, *i.e.*, jumps at t_1, t_2, t_3 in Figure 1(d). The update rules in CoPE are different from previous methods in that we adopt graph convolution-like aggregation rules to handle possibly concurrent interactions at a single time point. Previous methods process interactions one after another and heavily rely on the strict orderings of interactions. In addition to immediate impacts, an interaction also establishes a new message-passing channel between users and items. Then the model carries on continuous propagation on the updated graph until we observe the next interaction.

We train CoPE on interactions sequences with a temporal point process based loss [5]. Due to the large number of interactions in real applications, we resort to meta-learning techniques [25] to enable CoPE to adapt to recent interactions. Furthermore, considering the difficulties in training neural ODEs [4], we also propose a fast approximation to the closed-form solution to the ODE we used, which dramatically improves the speed of CGNNs.

Overall, the contributions of this paper are as follows: (1) We propose CoPE, which models continuous propagation and evolution on interaction graphs. (2) We use ODE based graph neural

networks as the continuous propagation and evolution unit. Instead of calling ODE solvers, we also propose a fast approximation to the closed-from solution to the ODEs. (3) We propose the discrete update unit to model immediate impacts of interactions. The graph convolution-like aggregation rules enable CoPE to handle concurrent interactions effectively. (4) To deal with long interaction sequences, we use meta-learning techniques to train CoPE so it can adapt to recent interactions. (5) We evaluate the proposed method on item recommendation, future interaction prediction and user state change prediction tasks and we show that CoPE outperforms baselines by 17.42%, 14.16% and 2.02% on average on these tasks respectively.

2 RELATED WORK

2.1 Temporal (Interaction) Graph Embedding

Representation learning on graphs has been widely studied since it can generate node embeddings for various downstream tasks. Different from methods like Deepwalk [28] and node2vec [10] that focus on static graphs, there are some recent works paid attention to temporal networks, *i.e.*, networks with timestamped edges. For example, HTNE [40] treats neighborhood formation sequences as Hawkes processes and learns node embeddings by maximum likelihood estimation (MLE). CTDNE [24] generates node embeddings by examining temporally increasing random walks. Interaction graphs, as a special type of temporal networks, have special properties for better node embedding. By modeling temporal dependency among node induced sequences, IGE [35, 36] could generate node embeddings with two multiplicative neural networks. TigeCMN [37] enhances this idea by incorporating memory networks. However, these methods fail to depict how nodes evolve in a temporal network, since they only assign a static embedding to each node at the end of training. They also have to be re-trained when new edges are established.

Instead of learning static node embeddings, DeepCoevolve [5] and JODIE [19] try to generate node embedding trajectories, *i.e.*, node embeddings as functions of time. DeepCoevolve, JODIE and the proposed method CoPE are all based on the observation that users and items influence each other and co-evolve over time. DeepCoevolve formalizes interaction sequences as mutually exciting point processes with two intertwined recurrent neural networks (RNNs), and train the model with MLE. But DeepCoevolve will not update node embeddings until an interaction occurs resulting in piecewise constant embeddings as shown in Figure 1(b). To tackle the issue, JODIE introduces a projection operation to estimate user embeddings at any time as shown in Figure 1(c). The estimated user embeddings are directly used to predict the next interaction. However, as we point out in Section 1, DeepCoevolve and JODIE fail to generate fully-temporal embeddings and overlook information propagation on interaction graphs. We make a more detailed comparison in Section 4.5.

2.2 Sequential Models

Some sequential modeling methods also explore dynamics in interactions. Sequential recommendations try to predict what a user would like to interact with given his/her historical interaction records. In this way, advanced architectures like convolutional/recurrent neural networks [12, 30], transformers [15, 21], *etc.*, can be utilized to exploit dynamic of a user within the session. But most existing methods are user-centric and neglect items evolution. RRN [31] introduces a pair of RNNs to model both user and item dynamics. Another shortcoming is that most methods like GRU4Rec [12], Caser [30] and SASRec [15] do not consider irregular time intervals between interactions. Some methods like T-LSTM [1] and RRN treat elapsed time as an extra feature in RNNs, but recent studies [6, 29] have shown that these types of RNNs are inferior to ODE based models when modeling irregular-sampled time series.

3 PRELIMINARIES

Given a user set $\mathcal{U}$ and an item set $\mathcal{I}$, the sequential interactions between users and items can be organized as an ordered set $\mathcal{E} = \{(u_k, i_k, t_k)\}_{k=1}^{n}$, where $u_k \in \mathcal{U}, i_k \in \mathcal{I}$ and $0 = t_0 \leq t_1 \leq t_2 \leq \cdots \leq t_n \leq T$. Each interaction may be associated with a vector $\boldsymbol{f}(u_k, i_k, t_k)$, if it is attributed. Without loss of generality, we can normalize the time range of sequential interactions into $[0, 1]$, and then we have $t_0 = t_1 = 0$ and $t_n = T = 1$.

These interactions can be represented by an interaction graph [35, 36, 38] as illustrated in Figure 1(a).

Definition 1 (Interaction Graph). *An interaction graph $\mathcal{G} = (\mathcal{U} \cup \mathcal{I}, \mathcal{E})$ is a user-item bipartite graph, where each edge $e = (u, i, t) \in \mathcal{E}$ represents an interaction between user $u \in \mathcal{U}$ and item $i \in \mathcal{I}$ happened at time t.*

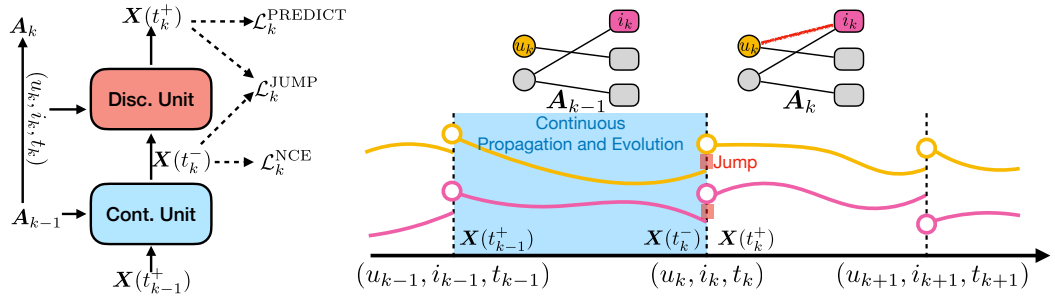
Definition 2 (Observable Graph). *Given an interaction graph $\mathcal{G} = (\mathcal{U} \cup \mathcal{I}, \mathcal{E})$ with the ordered edge set $\mathcal{E} = \{(u_k, i_k, t_k)\}_{k=1}^{n}$, the observable graph at time t is the subgraph with edges, i.e., interactions, happened before time t. The adjacency matrix of the observable graph at time $t \in (t_k, t_{k+1})$ is denoted by $\boldsymbol{A}_k = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{B}_k \\ \boldsymbol{B}_k^\top & \boldsymbol{0} \end{bmatrix}$, where $\boldsymbol{B}_k \in \Re^{|\mathcal{U}| \times |\mathcal{I}|}$ is the bi-adjacency matrix. The element $\boldsymbol{B}_{k,ui}$ denotes the number of interactions between u and i before time t_{k+1}, i.e., $\boldsymbol{B}_{k,ui} = |\{(u', i', t') \in \mathcal{E} | u' = u \wedge i' = i \wedge t' < t_{k+1}\}|$.*

Definition 3 (Temporal Embedding of Interaction Graph). *Given an interaction graph $\mathcal{G} = (\mathcal{U} \cup \mathcal{I}, \mathcal{E})$, the goal of temporal embedding is to learn a function $\boldsymbol{x} : (\mathcal{U} \cup \mathcal{I}) \times [0, T] \rightarrow \Re^d$ that reflects the continuous evolution of users and items over time. $\boldsymbol{x}(u, t)$ and $\boldsymbol{x}(i, t)$ are the d-dimensional embeddings of user u and item i at time t respectively.*

We assume all vectors are row vectors throughout the paper. We use $\boldsymbol{X}_{\mathcal{U}}(t) \in \Re^{|\mathcal{U}| \times d}$ and $\boldsymbol{X}_{\mathcal{I}}(t) \in \Re^{|\mathcal{I}| \times d}$ to denote stacked representations of users and items respectively. And $\boldsymbol{X}(t) \in \Re^{|\mathcal{U} \cup \mathcal{I}| \times d}$ denotes representations of all nodes in the interaction graph.

4 METHODOLOGY

In this section, we introduce our proposed method CoPE (**Co**ntinuous **P**ropagation and **E**volution). CoPE is composed of two basic units: the continuous propagation and evolution unit Cont_Unit$(\cdot)$ and the discrete update unit Disc_Unit$(\cdot)$. The former unit is utilized to simulate continuous propagation on interaction graphs and evolution of nodes between two consecutive interactions. The latter is to update node representations to reflect the direct impacts of



(a) The recurrent unit and the losses. **(b) An example forward pass.**

Figure 2: Illustrations of CoPE.

interactions. Thus we can obtain the following recurrent form:

$$\begin{cases} \boldsymbol{X}(0^+) = \boldsymbol{E}, \\ \boldsymbol{X}(t_k^-) = \text{Cont\_Unit}(\boldsymbol{X}(t_{k-1}^+), t_{k-1}, t_k) & k = 1, \dots, n, \\ \boldsymbol{X}(t_k^+) = \text{Disc\_Unit}(\boldsymbol{X}(t_k^-), u_k, i_k) & k = 1, \dots, n, \end{cases}$$

where $\boldsymbol{E} \in \Re^{|\mathcal{U} \cup \mathcal{V}| \times d}$ is the initial representations of all nodes (users and items), and $\boldsymbol{X}(t^-)$ and $\boldsymbol{X}(t^+)$ denote node embeddings before and after the jump at t. The initial representations $\boldsymbol{E}$ can be obtained from an embedding lookup table. We illustrate this recurrent unit in Figure 2(a).

4.1 Continuous Propagation and Evolution Unit

Assume that we have observed interactions $(u_1, i_1, t_1), \dots, (u_k, i_k, t_k)$ till time t_k. The goal of the continuous propagation and evolution unit is to answer the question: what $\boldsymbol{X}(t)$ $(t \in (t_k, t_{k+1}))$ will be given $\boldsymbol{X}(t_k^+)$ before we observe the next interaction.

To define continuous latent states between observations, early attempts [3] tried to concatenate timestamps to the features and feed them into neural networks. Another line of work [1, 39] assumed latent states are following a simple exponential decay. Recent works have shown that these methods are less effective than ordinary differential equations (ODEs) based methods [6, 29].

By describing the change rates of functions, ODEs can naturally reflect values at any time. So in this paper, we assume that evolution of nodes ,*i.e.*, users and items, follows an ODE: $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X}(t) = h_k(\boldsymbol{X}(t), t)$, $t \in (t_k, t_{k+1})$, where $h_k : \Re^{|\mathcal{U} \cup \mathcal{V}| \times d} \times [0, T] \to \Re^{|\mathcal{U} \cup \mathcal{V}| \times d}$ is a function depending on $\boldsymbol{X}(t)$. We can obtain node representations at time $t \in (t_k, t_{k+1})$ with $\boldsymbol{X}(t) = \boldsymbol{X}(t_k^+) + \int_{t_k}^{t} h \, \mathrm{d}\tau$.

Now we need to define the function h_k. Recall that nodes evolve as information propagates on the graph. So the function h_k also need to take care of continuous propagation. A natural way to model information flow on graphs is to use graph neural networks (GNNs) [16]. By letting h be a GNN-based function, we simultaneously model continuous propagation and evolution on the interaction graph. Specifically, we adopt the continuous graph neural networks (CGNNs) [33].

Let us define the propagation matrix

$$\boldsymbol{L}_k = \frac{\alpha}{2}\left(\boldsymbol{I} + \boldsymbol{D}_k^{-\frac{1}{2}} \boldsymbol{A}_k \boldsymbol{D}_k^{-\frac{1}{2}}\right), \tag{1}$$

where $\boldsymbol{A}_k$ is the adjacency matrix of the observable graph at time $t \in (t_k, t_{k+1})$, $\boldsymbol{D}_k$ is the degree matrix and $\alpha \in (0, 1)$ is a parameter controlling the spectral radius of $\boldsymbol{L}_k$. From another view, α controls how quickly a center node affects its neighbors. Thus it is plausible to set α for each node, *i.e.*, using a diagonal matrix, and learn its value from data. We use this strategy in the experiments, but we still use Eq. 1 to facilitate the discussion.

Then we have the following CGNN [33]:

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X}(t) = (\boldsymbol{L}_k - \boldsymbol{I})\boldsymbol{X}(t) + \boldsymbol{E}, \ t \in (t_k, t_{k+1}), \tag{2}$$

where $\boldsymbol{I}$ is the identity matrix. Compared to traditional GNNs, CGNNs have the propagation rule free from feature transformation and nonlinear activation, which have been shown less important than neighborhood aggregation [11, 32]. Moreover, the $\boldsymbol{E}$ in Eq. 2 prevents CGNNs from over-smoothing [17, 33].

4.2 Discrete Update Unit

As pointed out by previous studies, an interaction (u, i, t) would make an immediate impact on the involved user u and item i. So we need to update the embeddings of users who took action at time t:

$$\begin{aligned} \Delta \boldsymbol{x}(u, t) &= \sigma\Big(\boldsymbol{x}(u, t^-) \cdot \boldsymbol{W}_1 + \frac{1}{C_{u,t}} \sum_{i' \in \mathcal{N}(u,t)} \boldsymbol{x}(i', t^-) \cdot \boldsymbol{W}_2\Big), \\ \boldsymbol{x}(u, t^+) &= \boldsymbol{x}(u, t^-) + \Delta \boldsymbol{x}(u, t), \end{aligned} \tag{3}$$

where σ is the activation function, $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \Re^{d \times d}$ are weight matrices, $\mathcal{N}(u, t) = \{i' | (u', i', t') \in \mathcal{E} \text{ s.t. } u' = u \wedge t' = t\}$ and $C_{u,t} = |\mathcal{N}(u, t)|$. Depending on the time granularity, there may be multiple concurrent interactions at time t. So we need to aggregate information from all relevant interactions, *i.e.*, summation in Eq. 3.

For all users, the above update rule can be organized in a concise matrix form:

$$\begin{aligned} \Delta \boldsymbol{X}_{\mathcal{U}}(t) &= \sigma\left(\boldsymbol{X}_{\mathcal{U}}(t^-) \cdot \boldsymbol{W}_1 + \boldsymbol{D}_{\mathcal{U}}^{-1} \cdot \Delta \boldsymbol{B}_{\boldsymbol{k}} \cdot \boldsymbol{X}_{\mathcal{I}}(t^-) \cdot \boldsymbol{W}_2\right), \\ \boldsymbol{X}_{\mathcal{U}}(t^+) &= \boldsymbol{X}_{\mathcal{U}}(t^-) + \boldsymbol{M}_{\mathcal{U}}(t) \odot \Delta \boldsymbol{X}_{\mathcal{U}}(t), \end{aligned} \tag{4}$$

where $\Delta \boldsymbol{B}_k = \boldsymbol{B}_k - \boldsymbol{B}_{k-1}$ (here we assume $t = t_k$ for notation simplicity), $\boldsymbol{D}_{\mathcal{U}} \in \Re^{|\mathcal{U}| \times |\mathcal{U}|}$ is a diagonal matrix constructed from row summation of $\Delta \boldsymbol{B}_{\boldsymbol{k}}$, and $\boldsymbol{M}_{\mathcal{U}}(t) \in \{0, 1\}^{|\mathcal{U}| \times d}$ is a masking matrix. If u takes action at time t, then the corresponding row in $\boldsymbol{M}_{\mathcal{U}}(t)$ is filled with all ones, otherwise zeros.

Similarly, we can define the update rule for items:

$$\begin{aligned}\Delta \boldsymbol{X}_{\mathcal{I}}(t) &= \sigma\left(\boldsymbol{X}_{\mathcal{I}}(t^-)\cdot \boldsymbol{W}_3 + \boldsymbol{D}_{\mathcal{I}}^{-1}\cdot \Delta \boldsymbol{B}_{\boldsymbol{k}}^{\top}\cdot \boldsymbol{X}_{\mathcal{U}}(t^-)\cdot \boldsymbol{W}_4\right),\\ \boldsymbol{X}_{\mathcal{I}}(t^+) &= \boldsymbol{X}_{\mathcal{I}}(t^-) + \boldsymbol{M}_{\mathcal{I}}(t)\odot \Delta \boldsymbol{X}_{\mathcal{I}}(t).\end{aligned} \tag{5}$$

Note that the diagonal matrix $\boldsymbol{D}_{\mathcal{I}} \in \Re^{|\mathcal{I}|\times|\mathcal{I}|}$ is computed from the column summation of $\Delta \boldsymbol{B}_{\boldsymbol{k}}$. Eq. 4 and Eq. 5 can be regarded as graph convolution layers with residual connections, but defined on the difference of two consecutive graphs.

If interactions are attributed, we also can inject attributes into the above update rules, for example, by modifying Eq. 3:

$$\Delta \boldsymbol{x}(u,t) = \sigma\Big(\boldsymbol{x}(u,t^-)\cdot \boldsymbol{W}_1 + \frac{1}{C_{u,t}}\sum_{i'}\left(\boldsymbol{x}(i',t^-)\cdot \boldsymbol{W}_2 + \boldsymbol{f}(u,i',t)\cdot \boldsymbol{W}_5\right)\Big),$$

where $\boldsymbol{f}(u,i',t)$ is the attribute vector associated with the edge (u,i',t), and $\boldsymbol{W}_5$ is another weight matrix with appropriate dimensions.

To summarize, CoPE defines the following hybrid dynamic system:

$$\begin{cases}\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X}(t) = (\boldsymbol{L}_k - \boldsymbol{I})\boldsymbol{X}(t) + \boldsymbol{E} & t\in(t_k, t_{k+1}), k = 1,\dots,n-1,\\ \boldsymbol{X}(t_k^+) = \mathrm{Disc\_Unit}(\boldsymbol{X}(t_k^-)) & k = 1,\dots,n,\\ \boldsymbol{X}(0^+) = \boldsymbol{E}.\end{cases} \tag{6}$$

We illustrate the forward pass of CoPE in Figure 2(b).

4.3 Training CoPE with Meta-Learning

To train CoPE on sequential interactions, we follow previous methods [5, 40] and treat them as a multi-dimensional temporal point process. We define the intensity function for each user-item pair as

$$\lambda(u,i,t) = \exp\left(\mathrm{FC}_{\mathcal{U}}\left(\boldsymbol{x}(u,t^-)\|\boldsymbol{e}(u)\right)\cdot \mathrm{FC}_{\mathcal{I}}\left(\boldsymbol{x}(i,t^-)\|\boldsymbol{e}(i)\right)^{\top}\right), \tag{7}$$

where $\mathrm{FC}_{\mathcal{U}}$ and $\mathrm{FC}_{\mathcal{I}}$ are fully-connected layers for users and items respectively, and $\|$ denotes the concatenation operation. Note that in Eq. 7 we make use of the initial embedding $\boldsymbol{E}$ to capture static features of users and items. The negative log-likelihood function is given by

$$\mathcal{L}^{\mathrm{MLE}} = -\sum_{k=1}^{n}\log\lambda(u_k,i_k,t_k) + \sum_u\sum_i\int_0^T \lambda(u,i,t)\,\mathrm{d}t, \tag{8}$$

By minimizing Eq. 8, we simultaneously maximize the probability of happened interactions, and minimize the probability of non-events [23].

However, the second term in Eq. 8 is hard to compute. We instead use the noise contrastive estimation following [5, 40], where the step loss at t_k is defined as:

$$\mathcal{L}_k^{\mathrm{NCE}} = -\log\lambda(u_k,i_k,t_k) + \log\sum_{(u,i)\in\mathcal{S}}\lambda(u,i,t_k),$$

where $\mathcal{S}\subset|\mathcal{U}|\times|\mathcal{I}|$ is the set of negative pairs.

In some cases, we are required to predict whether a node would change its state after taking action [19]. So we can compute the prediction loss $\mathcal{L}_k^{\mathrm{PREDICT}}$ based on $\boldsymbol{X}(t_k^+)$. This loss can be defined according to properties of the downstream task. For example, we use binary cross-entropy loss in the Experiment 3 (Section 5.3).

Algorithm 1 Meta-learning of CoPE (one epoch).

Require: Interactions $(u_1,i_1,t_1),\dots,(u_n,i_n,t_n)$, model parameters $\boldsymbol{\Theta}$ (including $\boldsymbol{E}$), meta-learning step size ϵ

```
Θ̃ ← Θ
Split interactions into chunks
for i = 1, 2, ... do
    for j = 1, ..., inner_steps do
        Compute the i-th chunk loss L^TBPTT with Θ̃
        Θ̃ ← Adam(L^TBPTT)
    end for
    if ml_steps divides i then
        Θ ← Θ + ε(Θ̃ − Θ)
        Θ̃ ← Θ
    end if
end for
Return Θ
```

Recall that we update the node embeddings when we observe new interactions. To avoid dramatically distorting the embedding manifold and make the training process smooth, we introduce the jump loss as a regularization:

$$\mathcal{L}_k^{\mathrm{JUMP}} = \|\boldsymbol{M}_{\mathcal{U}}(t_k)\odot\Delta\boldsymbol{X}_{\mathcal{U}}(t_k)\|_F^2 + \|\boldsymbol{M}_{\mathcal{I}}(t_k)\odot\Delta\boldsymbol{X}_{\mathcal{I}}(t_k)\|_F^2, \tag{9}$$

where $\|\cdot\|_F^2$ is the squared Frobenius norm.

Now we have the total loss as $\mathcal{L} = \sum_k \mathcal{L}_k^{\mathrm{NCE}} + \mathcal{L}_k^{\mathrm{PREDICT}} + \beta\mathcal{L}_k^{\mathrm{JUMP}}$, regularized by the coefficient $\beta \geq 0$. Note that the term $\mathcal{L}_k^{\mathrm{PREDICT}}$ is dropped if no node-level downstream tasks are involved, *e.g.*, Experiments 1&2 in Section 5. From Figure 2(a) we can observe that these three losses come from different states of nodes.

To train the model on the sequence of interactions, we adopt the truncated back propagation through time (BPTT). We split the sequence of interactions into small chunks and each contains *tbptt_len* interactions. Given a chunk ending with (u_k,i_k,t_k), we compute the total loss within the chunk and do back propagation. Then we pass $X(t_k^+)$ as an initial value to the next chunk, and the gradients in the next chunk won't back propagate through $X(t)(t<t_k)$.

Compared to sentences in natural language processing, sequences of interactions are much longer. For example, in LastFM dataset (see Table 3), there are 1,293,103 interactions constituting a single sequence. This brings difficulties in training with truncated BPTT as parameters' values and gradients would oscillate more frequently across chunks. When encounter with a new chunk, the model parameters are required to adapt to the chunk quickly. So we novelly treat chunks as different tasks and train CoPE with the help of meta-learning for fast adaptation. We regard consecutive *ml_steps* chunks as a task, and update model parameters $\boldsymbol{\Theta} = \{\boldsymbol{E}, \boldsymbol{W}_1, \boldsymbol{W}_2, \dots\}$ according to Reptile [25]. This also empowers us to adapt to new interactions happened at inference phase. The meta-learning of Cope is illustrated in Algorithm 1.

4.4 Fast Approximation of CGNNs

From Eq. 6 we can see that we need to repeatedly solve neural ODEs with the following form:

Table 1: Comparison among different methods.

	DeepCoevolve [5]	JODIE [19]	CoPE [This paper]
Embeddings at time $t \in (t_k, t_{k+1})$	the same as $\boldsymbol{X}(t_k^+)$	a linear function of $\boldsymbol{X}(t_k^+)$	defined by an ODE
Information Propagation	no	no	yes
User embeddings at all times	no	yes (by linear projections)	yes
Item embeddings at all times	no	no	yes
Computation of $\boldsymbol{X}(t_k^+)$	$(\boldsymbol{X}(t_{k-1}^+), t_k - t_{k-1}) \to \boldsymbol{X}(t_k^+)$	$(\boldsymbol{X}(t_{k-1}^+), t_k - t_{k-1}) \to \boldsymbol{X}(t_k^+)$	$\boldsymbol{X}(t_{k-1}^+) \to \boldsymbol{X}(t_k^-) \to \boldsymbol{X}(t_k^+)$
Symmetry	symmetric	user-centric	symmetric
Loss function	point process based	ℓ_2 distance between embeddings	point process based
Intensity of (u, i, t)	$\exp(\boldsymbol{x}(u, t') \cdot \boldsymbol{x}(i, t')^\top)(t - t')$	-	$\exp(\boldsymbol{x}(u, t^-) \cdot \boldsymbol{x}(i, t^-)^\top)$
Evaluation phase adaptation	no	yes	yes
Concurrent interactions	sequential	sequential	aggregation

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X}(t) = (\boldsymbol{L} - \boldsymbol{I})\boldsymbol{X}(t) + \boldsymbol{E}, \\ \boldsymbol{X}(0) = \boldsymbol{Y}, \end{cases} \tag{10}$$

where we drop all subscripts and superscripts for simplicity, and assume the initial value is given at $t = 0$.

To solve this neural ODE and meantime enable gradients flow, researchers have developed the solver with the adjoint sensitivity method [4]. But solving ODEs and training neural networks with ODEs embedded are both tricky and time-consuming [8]. In this paper, we try to use the analytic solution to the Eq. 10:

$$\boldsymbol{X}(t) = (\boldsymbol{L} - \boldsymbol{I})^{-1} \cdot \left(e^{(\boldsymbol{L}-\boldsymbol{I})t} - \boldsymbol{I}\right) \cdot \boldsymbol{E} + e^{(\boldsymbol{L}-\boldsymbol{I})t} \cdot \boldsymbol{Y}. \tag{11}$$

The difficulties in evaluating the above formula is how to compute the matrix inverse $(\boldsymbol{L}-\boldsymbol{I})^{-1}$ and the matrix exponential $e^{(\boldsymbol{L}-\boldsymbol{I})t}$ efficiently. Luckily, from Eq. 1 we can notice that the spectral radius of $\boldsymbol{L}$ is $\alpha < 1$, which enables accurate approximations to these operations.

For the matrix inverse, we resort to the Neumann series:

$$(\boldsymbol{I} - \boldsymbol{L})^{-1} = \sum_{r=0}^{\infty} \boldsymbol{L}^r \approx \sum_{r=0}^{R_{\text{inv}}} \boldsymbol{L}^r \tag{12}$$

Since we have $\lim_{r\to\infty} \boldsymbol{L}^r = \boldsymbol{0}$, the series converges. We use the truncated series as an approximation.

For the matrix exponential, we notice that $e^{(\boldsymbol{L}-\boldsymbol{I})t} \cdot \boldsymbol{E} = e^{\boldsymbol{L}t} \cdot (e^{-t}\boldsymbol{E})$ and $e^{\boldsymbol{L}t}$ is analogous to the graph heat kernel [7, 34] with the scale parameter t. Thus we use the Chebyshev polynomials to approximate it:

$$e^{\boldsymbol{L}t} = \left[e^{\tilde{\boldsymbol{L}}t}\right]^{\lceil t \rceil} \approx \left[\sum_{r=0}^{R_{\text{exp}}} c_r \mathcal{T}_r\left(\tilde{\boldsymbol{L}}\right)\right]^{\lceil t \rceil}, \tag{13}$$

where $\mathcal{T}_r(\tilde{\boldsymbol{L}}) = \begin{cases} \boldsymbol{I} & r = 0 \\ \tilde{\boldsymbol{L}} & r = 1 \\ 2\tilde{\boldsymbol{L}}\mathcal{T}_{r-1}(\tilde{\boldsymbol{L}}) - \mathcal{T}_{r-2}(\tilde{\boldsymbol{L}}) & r > 1 \end{cases}$ are the Chebyshev polynomials. We scale the matrix $\tilde{\boldsymbol{L}} = \frac{\boldsymbol{L}t}{\lceil t \rceil}$ to meet the domain where Chebyshev polynomials are defined. The coefficients can be computed by the Bessel function of the first kind $J_k(\cdot)$ as $c_0 = J_0(\mathrm{i}), c_r = 2 \cdot \mathrm{i}^r \cdot J_r(-\mathrm{i})(k > 0)$, where i is the imaginary unit.

The procedure to approximate Eq. 11 is concluded as follows:

$$\begin{aligned} &\tilde{\boldsymbol{E}} \| \tilde{\boldsymbol{Y}} = e^{-t} \cdot (\boldsymbol{E} \| \boldsymbol{Y}), \\ &\tilde{\boldsymbol{E}} \| \tilde{\boldsymbol{Y}} \leftarrow \sum c_r \mathcal{T}_r(\tilde{\boldsymbol{L}}) \cdot (\tilde{\boldsymbol{E}} \| \tilde{\boldsymbol{Y}}), \quad \text{repeat } \lceil t \rceil \text{ times}, \\ &\hat{\boldsymbol{E}} = \sum \boldsymbol{L}^r \cdot (\boldsymbol{E} - \tilde{\boldsymbol{E}}), \\ &\boldsymbol{X}(t) = \hat{\boldsymbol{E}} + \tilde{\boldsymbol{Y}}. \end{aligned}$$

Thanks to the recursive forms in Eq. 12 and Eq. 13, only sparse-dense matrix multiplication is involved, and the procedure is analogous to a $(R_{\text{inv}} + \lceil t \rceil \cdot R_{\text{exp}})$-layered graph neural network. We simply let $R_{\text{inv}} = 10, R_{\text{exp}} = 2$ throughout the experiments. We explain why and show speedup in Section 5.6. Note that we have $\lceil t \rceil = 1$ when we normalize the range of interactions into $[0, 1]$.

4.5 Comparison among DeepCoevolve, JODIE and CoPE

In Table 1 we make a comparison against state-of-the-art interaction graph embedding algorithms, DeepCoevolve and JODIE. From the table and Figure 1 we can observe that only CoPE can model continuous propagation and evolution on interaction graphs, and thus can generate users and items embeddings with sophisticated evolution patterns.

After observing interaction (u_k, i_k, t_k), DeepCoevolve and JODIE update node embeddings by the recurrent neural networks, whose inputs are node embeddings at the last time step $\boldsymbol{X}(t_{k-1}^-)$ and the elapsed time $t_k - t_{k-1}$. CoPE instead generates $\boldsymbol{X}(t_k^-)$ first by simulating continuous propagation and evolution, and then makes a discrete update and produces $\boldsymbol{X}(t_k^+)$. This process is more interpretable and free from modeling elapsed time.

DeepCoevolve and CoPE treat user nodes and item nodes equally and define symmetric models. But JODIE is a user-centric model: the linear projection operations are only utilized for users, and the loss function is defined as the ℓ_2 distance between the item embedding predicted from user embeddings and the true next item's embedding.

Speaking of the loss functions, CoPE follows DeepCoevolve and adopts temporal point process based loss functions, which explain why interactions happen, but also why other interactions did not happen, *i.e.*, non-events. The ℓ_2 loss in JODIE is unable to explain non-events. But intensity functions used by DeepCoevolve and

Table 2: Statistics of Recommendation Datasets.

	# Users	# Items	# Interactions	# Unique Times
ML-100K	943	1,349	99,287	49,119
ML-1M	6,040	3,416	999,661	458,254
Yoochoose	33,670	3,667	211,851	41,374
Garden	1,686	962	13,272	1,888
Video	5,130	1,685	37,126	1,946
Game	24,303	10,672	231,780	5,302

Table 3: Statistics of Interaction Datasets.

	# Users	# Items	# Interactions	# Unique Times
Wikipedia	8,227	1,000	157,474	152,757
LastFM	980	1,000	1,293,103	1,283,614
MOOC	7,047	97	411,749	345,600

CoPE differ. Since DeepCoevolve cannot generate representations just before time t, to express the intensity of new interaction (u, i, t) we have to make use of previous representations at time t'. Here t' is the last time point where either u or i took action. An extra linear decay term is also involved to model elapsed times. But the manually defined decay function may be inappropriate in some cases [6, 29]. In our proposed CoPE, we can generate node representations at all times with an ODE. So we can directly model the intensity function with $\boldsymbol{X}(t^-)$ and we are free from designing decay functions. Here we suppress $\mathrm{FC}(\cdot)$ and $\boldsymbol{E}$ parts in Eq. 7 for simplicity.

Considering the length of sequential interactions, JODIE also updates model parameters during the evaluation phase, *i.e.*, test-time training. After making the prediction and logging the metrics of some interaction, JODIE treats it as an seen interaction, then does one step gradient descent with the corresponding step loss. However, it is possible JODIE forgets the previous interactions during the test-time training process, i.e., catastrophic forgetting [14]. In CoPE, we train the model with meta-learning (Algorithm 1) and do fast adaptation, *i.e.*, gradient descent, at evaluation phase. Now CoPE's training behaviors at training time and testing time would be more consistent than JODIE's, which also alleviates catastrophic forgetting phenomena [14, 25].

Last but not least, DeepCoevolve and JODIE process interactions sequentially with a pair of recurrent neural networks. So when facing too many concurrent interactions, they would perform badly. Our proposed CoPE utilizes graph convolution-like aggregation rules Eq. 4 and Eq. 5 to update embeddings and can naturally handle concurrent interactions. We show the superiority of this discrete update unit with experiments.

5 EXPERIMENTS

In this section, we prove the effectiveness of the proposed CoPE method by three groups of experiments.

The codes of CoPE are available at https://github.com/yzhang1918/cikm2021cope. Datasets used in this paper are publicly available, and we also attach download links in the repository. Statistics of datasets are shown in Table 2 and Table 3.

Setting: We use the same experimental setting as JODIE's [19]. We split the data by time: the first 80% interactions are for training, the following 10% data for validation, and the next 10% data for testing. For the third experiment, the split ratios are 60%:20%:20%. The dimension of user/item embeddings is 128. All algorithms are run for 50 epochs, and we report results on the testing set when we obtain the best performance on the validation set. For Experiment 1 and Experiment 2, we report MRR (mean reciprocal rank) and Recall@10 scores. For Experiment 3, we report AUROC scores.

5.1 Experiment 1: Item Recommendation

In this group of experiments we show whether our proposed CoPE can accurately predict what items a user would interact with, which is a typical application of interaction graphs.

We use four publicly available datasets. MovieLens[1] is a widely used movie rating dataset. We use two versions of MovieLens: ML-100K and ML-1M. Yoochoose[2] is the user buying dataset. Amazon[3] contains ratings on the e-commerce platform Amazon. We adopt 3 sub-datasets of Amazon: Garden, Video and Game. We follow the same preprocessing pipeline in the literature [15, 30] that users and items with fewer than 5 observations are discarded. The statistical information is shown in Table 2 in Appendix. Please note that there are many concurrent interactions in these datasets.

We compare CoPE with the following baselines: (1) LightGCN [11] is the state-of-the-art collaborative filtering algorithm based on GNNs. This method ignores time information. (2) Time-LSTM [39] and (3) RRN [31] are sequential modeling algorithms that can generate temporal user embeddings. We skip other methods like Caser [30], GRU4Rec [12], SASRec [15] since they cannot handle irregular time intervals between interactions. (4) DeepCoevolve [5] and (5) JODIE [19] are state-of-the-art interaction graph embedding algorithms that can generate temporal user and item embeddings. We disable the test-time training trick used in JODIE, and denote this variation as JODIE *. (6) CoPE trained with the standard truncated BPTT without meta-learning is denoted by CoPE *. Since other comparing methods would not update model parameters at evaluation phase, we disable test-time training of JODIE and fast adaptation of CoPE for a fair comparison.

The experimental results are shown in Table 4. Clearly, our CoPE outperforms all baseline methods consistently. On average, CoPE gains 17.58% improvements of MRR and 17.26% improvements of Recall@10 against the best baselines. LightGCN performs worst on most datasets. This is due to the way we split the sequences of interactions: splitting by time results in many unseen users and items in the test set. LightGCN, based on static embeddings, has no knowledge of these nodes. One possible reason why the remaining methods are inferior to our proposal is that these methods process interactions one after another, but from Table 2 we can see that there are many concurrent interactions. So there are no strict orderings in these datasets. Our method uses graph convolution-like update rules to simultaneously update all nodes' embeddings and thus has a better result.

[1] https://grouplens.org/datasets/movielens/
[2] http://2015.recsyschallenge.com/challenge.html
[3] http://jmcauley.ucsd.edu/data/amazon/index.html

Table 4: Results on recommendation.

	Garden		Video		Game	
	MRR	Recall@10	MRR	Recall@10	MRR	Recall@10
LightGCN	0.025	0.087	0.019	0.036	0.015	0.026
Time-LSTM	0.038	0.134	0.028	0.044	0.014	0.020
RRN	0.072	0.152	0.033	0.068	0.018	0.029
DeepCoevolve	0.046	0.121	0.023	0.050	0.013	0.027
JODIE *	0.049	0.127	0.037	0.078	0.021	0.035
CoPE *	**0.081**	**0.192**	**0.048**	**0.088**	**0.026**	**0.047**
% improvement of CoPE	12.50%	26.32%	29.73%	12.82%	23.81%	34.29%
	ML100K		ML1M		Yoochoosebuy	
	MRR	Recall@10	MRR	Recall@10	$_{10\times}$MRR	$_{10\times}$Recall@10
LightGCN	0.012	0.025	0.013	0.029	0.055	0.091
Time-LSTM	0.022	0.058	0.018	0.033	0.073	0.124
RRN	0.032	0.065	0.023	0.043	0.086	0.127
DeepCoevolve	0.029	0.069	0.018	0.030	0.054	0.106
JODIE *	0.034	0.074	0.020	0.035	0.095	0.179
CoPE *	**0.038**	**0.081**	**0.025**	**0.049**	**0.113**	**0.191**
% improvement of CoPE	11.76%	9.46%	8.70%	13.95%	18.95%	6.70%

Table 5: Results on future interaction prediction.

	Wikipedia		LastFM	
	MRR	Recall@10	MRR	Recall@10
Time-LSTM	0.247	0.342	0.068	0.137
RRN	0.522	0.617	0.089	0.182
LatentCross	0.424	0.481	0.148	0.227
CTDNE	0.035	0.056	0.010	0.010
DeepCoevolve	0.515	0.563	0.019	0.039
JODIE	0.746	0.822	0.195	0.307
CoPE	**0.750**	**0.890**	**0.200**	**0.446**
% improvement of CoPE	0.54%	8.27%	2.56%	45.28%

5.2 Experiment 2: Future Interaction Prediction

The second task is similar to the previous one, but focuses on more general interactions between users and objects.

We use datasets[4] released by the authors of JODIE: (1) Wikipedia dataset contains one month of edits on Wikipedia pages. (2) LastFM is a dataset containing one month of listening history of 1000 users. The statistical information is shown in Table 3 in Appendix. Wikipedia is with attributed interactions. Compared to recommendation datasets, these datasets have denser connections between users and items, more repeated interactions but less concurrence.

We directly report results of baselines in JODIE's paper since we use exactly the same setting. JODIE is compared with two more baselines: LatentCross [2], a sequential recommendation method, and CTDNE [24], a temporal network embedding method.

[4]http://snap.stanford.edu/jodie/

From Table 5 we can observe that CoPE achieves satisfying results. CoPE obtains more significant improvements with respect to Recall@10. Especially on LastFM CoPE surpasses the best baseline JODIE by 45.28%. Recall that MRR is a global metric while Recall@10 only consider the top-10 recommendations. This result implies that our method is prone to rank relevant items much higher. For detailed analysis on baseline methods, please refer to the paper of JODIE [19].

5.3 Experiment 3: User State Change Prediction

The third task requires models to predict whether a user would change his/her state after taking action, *i.e.*, being banned after making a post.

We use (1) Wikipedia dataset with extra labels indicating banned users and the corresponding interactions, where we have 217 bans in Wikipedia. (2) MOOC dataset consists of attributed interactions between students and online courses, among which 4,066 students drop-out courses after certain interactions.

Again, We reuse the results of baselines in JODIE's paper. From Table 6 we can see that our proposed CoPE outperforms all baselines. This proves CoPE can capture features of interactions and change the representations accordingly with the discrete update unit.

5.4 Ablation Study

We introduce the jump loss Eq. 9 to avoid distorting embeddings too much after discrete updates. In Table 7 we show the results of CoPE without this regularizer by letting $\beta = 0$, which are slightly inferior to CoPE with searched β. This proves the effectiveness of the jump loss.

To deal with long sequences of interactions, we resort to meta-learning techniques in Section 4.3. Now we test if training with meta-learning could help the model to adapt to recently observed

Table 6: Results on state change prediction. AUROC scores are reported.

	Wikipedia	MOOC
Time-LSTM	0.671	0.711
RRN	0.804	0.558
LatentCross	0.628	0.686
DeepCoevolve	0.663	0.671
JODIE	0.831	0.756
CoPE	**0.858**	**0.762**
% improvement of CoPE	3.25%	0.79%

Table 7: Ablation Study. Recall@10 scores are reported.

	Wikipedia	Garden	Video
CoPE	**0.890**	**0.313**	**0.254**
CoPE w/o jump loss	0.871	0.292	0.233
CoPE *	0.806	0.192	0.088
JODIE	0.822	0.258	0.178
JODIE *	0.699	0.127	0.078

interactions and improve the performance. In Table 7 we compare CoPE with CoPE *, and JODIE with JODIE*. Clearly, fast adaptation during evaluation phase could significantly boost the performance of CoPE. Though JODIE also gains a lot from test-time training, it lacks theoretical supports and may suffer catastrophic forgetting. Together from Table 4 and 5 we can note that even without meta-learning, CoPE * beats all comparing methods. This proves that our proposal itself is effective, and training with meta-learning could further maximizes its potential.

5.5 Model Robustness

Now we test if our proposal is robust with respect to the size of training data. We use the first 80% interactions as training data in previous experiments. Now we vary this percentage from 10% to 80% and report the corresponding results in Figure 3. We omit most baselines for their inferior performance. We can notice that both CoPE and JODIE are robust w.r.t the size of training data, while another comparing method, RRN, requires more data to achieve better performance. Nonetheless, CoPE is always taking the lead.

5.6 Efficiency of Approximate CGNNs

In Section 4.4, we propose an approximation of CGNNs, where we use the R_{inv}-order Neumann series and the R_{exp}-order Chebyshev polynomial. To reduce the interference of other components of CoPE and to evaluate the efficiency and effectiveness of approximate CGNNs solely, we compare original CGNNs and our approximate versions on the classic node classification task on the Cora dataset [16]. In Figure 4 we show the training time versus test accuracy curves for the original CGNN and our approximate versions with different order pairs $(R_{\text{inv}}, R_{\text{exp}})$. (Since we intend to show how accurate and how fast our approximation is, we do not tune hyper-parameters carefully.)

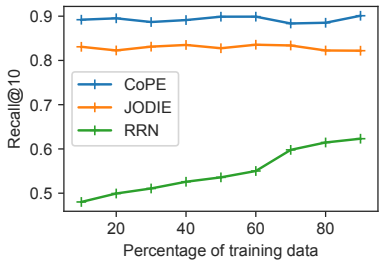


Figure 3: Robustness of CoPE w.r.t the training size. Recall@10 scores on Wikipedia are reported.

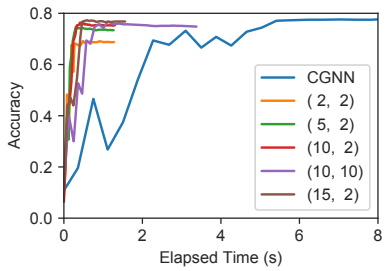


Figure 4: Time v.s. accuracy on Cora dataset. The blue line represents the original CGNN [33], while the other lines represent approximate CGNN with orders $(R_{\text{inv}}, R_{\text{exp}})$.

We can observe that our approximation can achieve near 10x speedup. The order of Neumann series for approximating the matrix inverse should be large, so we use $R_{\text{inv}} = 10$. The order of Chebyshev polynomials for approximating the matrix exponential is not sensitive, and $R_{\text{exp}} = 2$ gives satisfying results.

6 CONCLUSION

In this paper, we introduce our proposed method CoPE to generate temporal embeddings of users and items from sequential interactions. The continuous unit utilizes an ODE based GNN to model continuous evolution of nodes and information propagation on graphs. The discrete unit uses graph convolution-like update rules to model immediate impacts of interactions. We propose an efficient approximation of CGNNs and train the model with meta-learning. Extensive experiments prove the effectiveness of our proposal.

As for future work, we will consider how to extend our method to learning embeddings of general temporal networks. Besides, we will explore if introducing extra social networks or item similarity graphs is helpful.

ACKNOWLEDGMENTS

This work is funded in part by the National Natural Science Foundation of China Projects No. U1936213, No. U1636207, and the Shanghai Science and Technology Development Fund No. 19DZ1200802.

REFERENCES

[1] Inci M Baytas, Cao Xiao, Xi Zhang, Fei Wang, Anil K Jain, and Jiayu Zhou. 2017. Patient subtyping via time-aware lstm networks. In *KDD*. 65–74.

[2] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent cross: Making use of context in recurrent recommender systems. In *WSDM*. 46–54.

[3] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. 2018. Recurrent neural networks for multivariate time series with missing values. *Scientific reports* 8, 1 (2018), 1–12.

[4] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural ordinary differential equations. In *NeurIPS*. 6571–6583.

[5] Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. 2016. Deep coevolutionary network: Embedding user and item features for recommendation. *arXiv preprint arXiv:1609.03675* (2016).

[6] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. 2019. GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. In *NeurIPS*. 7379–7390.

[7] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. 2018. Learning structural node embeddings via diffusion wavelets. In *KDD*. 1320–1329.

[8] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. 2019. Augmented neural odes. In *NeurIPS*. 3140–3150.

[9] Ziwei Fan, Zhiwei Liu, Jiawei Zhang, Yun Xiong, Lei Zheng, and Philip S. Yu. 2021. Continuous-Time Sequential Recommendation with Temporal Graph Collaborative Transformer. In *CIKM*.

[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. 855–864.

[11] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, YongDong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*. 639–648.

[12] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *ICLR*.

[13] Jen-Hung Huang and Yi-Fen Chen. 2006. Herding in online product choice. *Psychology & Marketing* 23, 5 (2006), 413–428.

[14] Khurram Javed and Martha White. 2019. Meta-learning representations for continual learning. *arXiv preprint arXiv:1905.12588* (2019).

[15] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *ICDM*. 197–206.

[16] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[17] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*.

[18] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[19] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*. 1269–1278.

[20] Zhiwei Liu, Yingtong Dou, Philip S. Yu, Yutong Deng, and Hao Peng. 2020. Alleviating the Inconsistency Problem of Applying Graph Neural Network to Fraud Detection. In *SIGIR*.

[21] Zhiwei Liu, Ziwei Fan, Yu Wang, and Philip S. Yu. 2021. Augmenting Sequential Recommendation with Pseudo-Prior Items via Reversely Pre-training Transformer. In *SIGIR*.

[22] Zhiwei Liu, Mengting Wan, Stephen Guo, Kannan Achan, and Philip S Yu. 2020. BasConv: Aggregating Heterogeneous Interactions for Basket Recommendation with Graph Convolutional Neural Network. In *SDM*. 64–72.

[23] Hongyuan Mei and Jason M Eisner. 2017. The neural hawkes process: A neurally self-modulating multivariate point process. In *NeurIPS*. 6754–6764.

[24] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *WWW*. 969–976.

[25] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999* (2018).

[26] Nima Noorshams, Saurabh Verma, and Aude Hofleitner. 2020. TIES: Temporal Interaction Embeddings For Enhancing Social Media Integrity At Facebook. In *KDD*. 3128–3135.

[27] Martin Pavlovski, Jelena Gligorijevic, Ivan Stojkovic, Shubham Agrawal, Shabhareesh Komirishetty, Djordje Gligorijevic, Narayan Bhamidipati, and Zoran Obradovic. 2020. Time-Aware User Embeddings as a Service. In *KDD*. 3194–3202.

[28] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. 701–710.

[29] Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. 2019. Latent ordinary differential equations for irregularly-sampled time series. In *NeurIPS*. 5320–5330.

[30] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*. 565–573.

[31] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *WSDM*. 495–503.

[32] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*. 6861–6871.

[33] Louis-Pascal AC Xhonneux, Meng Qu, and Jian Tang. 2019. Continuous Graph Neural Networks. In *ICML*.

[34] Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. 2019. Graph wavelet neural network. In *ICLR*.

[35] Yao Zhang, Yun Xiong, Xiangnan Kong, Zhuang Niu, and Yangyong Zhu. 2019. IGE+: A Framework for Learning Node Embeddings in Interaction Graphs. *TKDE* (2019).

[36] Yao Zhang, Yun Xiong, Xiangnan Kong, and Yangyong Zhu. 2017. Learning node embeddings in interaction graphs. In *CIKM*. 397–406.

[37] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhao Li, and Can Wang. 2020. Learning Temporal Interaction Graph Embedding via Coupled Memory Networks. In *WWW*. 3049–3055.

[38] Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He. 2020. A data-driven graph generative model for temporal interaction networks. In *KDD*. 401–411.

[39] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. 2017. What to Do Next: Modeling User Behaviors by Time-LSTM.. In *IJCAI*, Vol. 17. 3602–3608.

[40] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding temporal network via neighborhood formation. In *KDD*. 2857–2866.