Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks

Srijan Kumar
Stanford University, USA and
Georgia Institute of Technology, USA
srijan@cs.stanford.edu

Xikun Zhang
University of Illinois,
Urbana-Champaign, USA
xikunz2@illinois.edu

Jure Leskovec
Stanford University, USA
jure@cs.stanford.edu

ABSTRACT

Modeling sequential interactions between users and items/products is crucial in domains such as e-commerce, social networking, and education. Representation learning presents an attractive opportunity to model the dynamic evolution of users and items, where each user/item can be embedded in a Euclidean space and its evolution can be modeled by an embedding trajectory in this space. However, existing dynamic embedding methods generate embeddings only when users take actions and do not explicitly model the future trajectory of the user/item in the embedding space. Here we propose *JODIE*, a coupled recurrent neural network model that learns the embedding trajectories of users and items. *JODIE* employs two recurrent neural networks to update the embedding of a user and an item at every interaction. Crucially, *JODIE* also models the future embedding trajectory of a user/item. To this end, it introduces a novel projection operator that learns to estimate the embedding of the user at any time in the future. These estimated embeddings are then used to predict future user-item interactions. To make the method scalable, we develop a *t-Batch* algorithm that creates time-consistent batches and leads to 9× faster training. We conduct six experiments to validate *JODIE* on two prediction tasks—future interaction prediction and state change prediction—using four real-world datasets. We show that *JODIE* outperforms six state-of-the-art algorithms in these tasks by at least 20% in predicting future interactions and 12% in state change prediction.

ACM Reference Format:
Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330895

1 INTRODUCTION

Users interact sequentially with items in many domains such as e-commerce (e.g., a customer purchasing an item) [48], education (a student enrolling in a MOOC course) [31], and social and collaborative platforms (a user posting in a group in Reddit) [19, 24]. The same user may interact with different items over a period of time and these interactions change over time [4, 5, 17, 21, 34, 37, 48]. These interactions create a *network of temporal interactions between users and items*. Figure 1 (left) shows an example network between users and items, with each interaction marked with a time stamp t_r and a feature vector $\boldsymbol{f_r}$ (such as the review text or the purchase amount). Accurate real-time recommendation of items and predicting change in the state of users are fundamental problems in these domains [5, 6, 22, 30, 36, 40, 43]. For instance, predicting when a student is likely to drop out of a MOOC course is important to develop early intervention measures [23, 46] and predicting when a user is likely to turn malicious on social platforms, like Reddit and Wikipedia, ensures platform integrity [9, 25, 26].

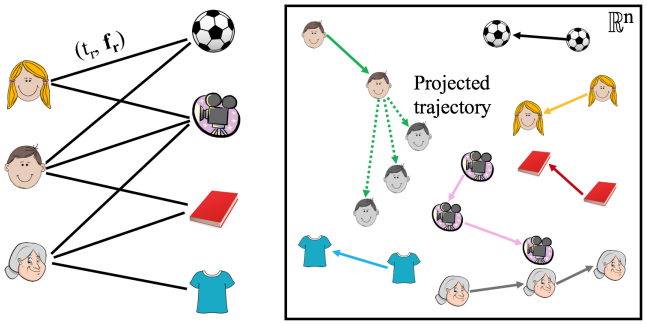

Figure 1: Left: a temporal interaction network of three users and four items. Each arrow represents an interaction with associated timestamp t and a feature vector f. Right: embedding trajectory of the users and items. We predict the future trajectory of users (the dotted line shown for one user) by training an embedding projection operator.

Representation learning, or learning low-dimensional embeddings of entities, is a powerful approach to represent the evolution of users' and items' properties [8, 11, 13, 14, 48, 50]. However, the recent methods that generate dynamic embeddings suffer from four fundamental challenges. **First,** a majority of the existing methods generate an embedding for a user only when she takes an action [8, 11, 47, 48, 50]. However, consider a user who makes a purchase today and its embedding is updated. The embedding will remain the same if it returns to the platform on the next day, a week later, or even a month later. As a result, the same predictions and recommendations will be made to her regardless of when she returns. However, a user's intent changes over time [10] and thus her embedding needs to be updated (projected) to the query time. The challenge here is how to accurately predict the embedding trajectories of users/items as time progresses. **Second**, entities have both stationary properties that do not change over time and time-evolving properties. Some existing methods [11, 44, 48] consider

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330895

only one of the two when generating embeddings. However, it is essential to consider both in a unified framework to leverage information at both scales. **Third**, many existing methods predict user-item interactions by scoring all items for each user [8, 11, 48, 50]. This has linear time complexity and is not practical in scenarios with millions of items. Instead, methods are required that can recommend items in near-constant time. **Fourth**, most models are trained by sequentially processing the interactions one at a time, so that the temporal dependencies between the interactions are maintained [11, 44, 48]. This prevents such models from scaling to datasets with millions of interactions. New methods are needed that can be trained with batches of data to generate embedding trajectories.

Present work. Here we present *JODIE* which learns to generate embedding trajectories of all users and items from temporal interactions[1]. The embedding trajectories of the example network are shown in Figure 1 (right). The embeddings of the user and item are updated when a user takes an action and a projection operator predicts the future embedding trajectory of the user.

Present work: *JODIE.* Each user and item has two embeddings: a static embedding and a dynamic embedding. The static embedding represents the entity's long-term stationary property, while the dynamic embedding represents time-varying property and is learned using the *JODIE* algorithm. Both embeddings are used to generate the trajectory. This enables *JODIE* to make predictions from both the stationary and time-varying properties of the user.

The *JODIE* model consists of two major components: an update operation and a projection operation.

The *update operation* of *JODIE* has two Recurrent Neural Networks (RNNs) to generate user and item embeddings. Crucially, the two RNNs are coupled to explicitly incorporate the interdependency between users and items. After each interaction, the user RNN updates the user embedding by using the embedding of the interacting item. Similarly, the item RNN uses the user embedding to update the item embedding. The model also has the ability to incorporate feature vectors from the interaction, for example, the text of a Reddit post. It should be noted that *JODIE* is easily extendable to multiple types of entities by training one RNN for each entity type. In the current work, we show how to apply *JODIE* to the case of bipartite interactions between users and items.

A major innovation of *JODIE* is that it also uses a *projection operation* that predicts the future embedding trajectory of the users. Intuitively, the embedding of a user will change slightly after a short time elapses since her previous interaction (with any item), while the embedding can change significantly after a long time elapses. As a result, *JODIE* trains a temporal attention layer to project the embedding of users after some time Δ elapses since its previous interaction. The projected user embedding is then used to predict the item that the user is most likely to interact with.

To predict the item that a user will interact with, an important design decision is to output the embedding of an item, instead of an interaction probability. Current methods generate a probability score of interaction between a user and an item, which takes linear time to find the most likely item because probability scores for all items have to be generated first. Instead, by directly generating the item embedding, we can recommend the item that is closest to the predicted item in the embedding space. This can be done efficiently in constant time using the locality sensitive hashing (LSH) techniques [27].

Present work: *t-Batch.* Most existing models learn embeddings from a sequence of interactions by processing one interaction after the other, in increasing order of time to maintain the temporal dependency among the interactions [11, 44, 49]. This makes such algorithms unscalable to real datasets with millions of interactions. Therefore, we create a batching algorithm, called *t-Batch*, to train *JODIE* by creating training batches of independent interactions such that the interactions in each batch can be processed in parallel. To do so, we iteratively select independent edge sets from the interaction network. In every batch, each user and item appears at most once and the temporally-sorted interactions of each user (and item) appear in monotonically increasing batches. Experimentally, we show that *t-Batch* makes *JODIE* 9.2× faster than its most similar dynamic embedding baselines.

Present work: Experiments. We conduct six experiments to evaluate the performance of *JODIE* on two tasks: predicting the next interaction of a user and predicting the change in state of users (when a user will be banned from social platforms and when a student will drop out from a MOOC course). We use four datasets from Reddit, Wikipedia, LastFM, and a MOOC course activity for our experiments. We compare *JODIE* with six state-of-the-art algorithms from three categories: deep recurrent recommender algorithms [8, 45, 52], temporal node embedding algorithm [33], and dynamic co-evolution models [11]. *JODIE* improves over the baseline algorithms on the interaction prediction task by at least 20% in terms of mean reciprocal rank and 12% in AUC on average for predicting user state change. We further show that *JODIE* is robust to the percentage of training data and the size of the embeddings.

Overall, in this paper, we make the following contributions:

- **Embedding algorithm:** We propose a coupled recurrent neural network model called *JODIE* to learn embedding trajectories of users and items. Crucially, *JODIE* also learns a projection operator to predict the embedding trajectory of users and predicts future interactions in constant time.
- **Batching algorithm:** We propose the *t-Batch* algorithm to create independent but temporally consistent training data batches that help to train *JODIE* 9.2× faster than the closest baseline.
- **Effectiveness:** *JODIE* outperforms six state-of-the-art algorithms in predicting future interactions and user state change predictions, by performing at least 20% better in predicting future interactions and 12% better on average in predicting user state change.

The code and datasets are available on the project website: `https://snap.stanford.edu/jodie`.

2 RELATED WORK

Here we discuss the research closest to our problem setting spanning three broad areas. Table 1 compares their differences.

Deep recurrent recommender models. Several recent models employ recurrent neural networks (RNNs) and variants (LSTMs and GRUs) to build recommender systems. RRN [45] uses RNNs to generate dynamic user and item embeddings from rating networks. Recent methods, such as Time-LSTM [52] and LatentCross [8] learn

[1] *JODIE* stands for Joint Dynamic User-Item Embeddings.

Table 1: Table comparing the desired properties of the existing algorithms and our proposed *JODIE* algorithm. *JODIE* satisfies all the desirable properties.

Property	Deep recurrent models			Temporal network embedding		Co-evolution models [11]	Proposed model
	LSTM, Time-LSTM [52]	RRN [45]	LatentCross [8]	CTDNE [33]	IGE [49]		*JODIE*
Predict embedding trajectory							✔
Predict future item embedding							✔
Train using batches of data	✔	✔	✔	✔			✔

how to incorporate features into the embeddings. However, most of these methods suffer from two major shortcomings. First, they take the one-hot vector of the item as input to update the user embedding. This only incorporates the item id and ignores the item's current state. The second shortcoming is that some models, such as Time-LSTM and LatentCross, generate dynamic embeddings only for users and not for items.

JODIE overcomes these shortcomings by learning embeddings for both users and items using mutually-recursive RNNs. In doing so, *JODIE* outperforms these methods by at least 20% in predicting the next interaction and 12% on average in predicting user state change, while having comparable running time as these methods.

Dynamic co-evolution models. Methods that jointly learn representations of users and items have recently been developed using point-process modeling [41, 44] and RNN-based modeling [11]. The basic idea behind these models is similar to *JODIE* —user and item embeddings influence each other whenever they interact. However, the major difference between *JODIE* and these models are that *JODIE* trains a project operation to forecast the user embedding at any time, outputs item embeddings instead of interaction probability, and trains the model using batching. As a result, we observe that *JODIE* outperforms DeepCoevolve by at least 44.8% in predicting the next interaction and 14% in predicting state change. In addition, most of these existing models are not scalable because they process interactions in a sequential order to maintain temporal dependency. *JODIE* overcomes this limitation by creating efficient training data batches which makes *JODIE* 9× faster than these baselines.

Temporal network embedding models. Several models have recently been developed that generate embeddings for the nodes (users and items) in temporal networks. CTDNE [33] is a state-of-the-art algorithm that generates embeddings using temporally-increasing random walks, but it generates one final static embedding of the nodes. Similarly, IGE [49] generates one final embedding of users and items from interaction graphs. Therefore, both these methods (CTDNE and IGE) need to be re-run for every new edge to create dynamic embeddings. Another recent algorithm, DynamicTriad [50] learns dynamic embeddings but does not work on bipartite interaction networks as it requires the presence of triads. Other recent algorithms such as DDNE [29], DANE [28], DynGem [15], Zhu et al. [51], and Rahman et al. [38] learn embeddings from a sequence of graph snapshots, which is not applicable to our setting of continuous interaction data. Recent models such as NP-GLM

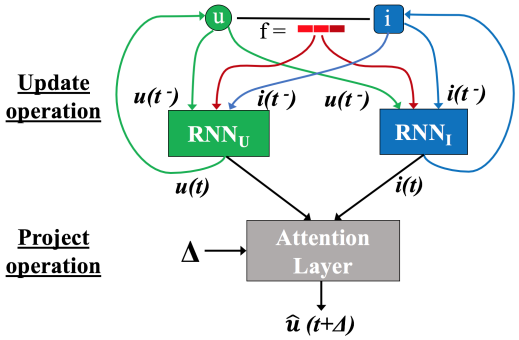

Figure 2: The *JODIE* model: After an interaction (u, i, t, f) between user u and item i, the dynamic embeddings of u and i are updated in the *update operation* with RNN_U and RNN_I, respectively. The *projection operation* predicts the user embedding at a future time $t + \Delta$.

model [39], DGNN [32], and DyRep [42] learn embeddings from persistent links between nodes, which do not exist in interaction networks as the edges represent instantaneous interactions.

Our proposed model, *JODIE* overcomes these shortcomings by generating and predicting the trajectories of users and items. In doing so, *JODIE* performs 4.4× better than CTDNE in predicting the next interaction, while having comparable running time.

3 JODIE: JOINT DYNAMIC USER-ITEM EMBEDDING MODEL

In this section, we propose *JODIE*, a method to learn embedding trajectories of users $\boldsymbol{u(t)} \in \mathbb{R}^n \; \forall u \in \mathcal{U}$ and items $\boldsymbol{i(t)} \in \mathbb{R}^n \; \forall i \in \mathcal{I}, \forall t \in [0, T]$ from an ordered sequence of temporal user-item interactions $S_r = (u_r, i_r, t_r, \boldsymbol{f_r})$. An interaction S_r happens between a user $u_r \in \mathcal{U}$ and an item $i_r \in \mathcal{I}$ at time $t_r \in \mathbb{R}^+, 0 < t_1 \leq t_2 \ldots \leq T$. Each interaction has an associated feature vector $\boldsymbol{f_r}$ (e.g., a vector representing the text of a post). Table 2 lists the symbols used. For ease of notation, we will drop the subscript r in the rest of the section.

Our proposed model, called *JODIE*, learns an embedding trajectory for users and items and is reminiscent of the popular Kalman Filtering algorithm [20].[2] *JODIE* uses the interactions to update the state of the interacting users and items via a trained *update operation*. *JODIE* trains a *projection operation* that uses the previous observed state and the elapsed time to predict the future embedding of the user. When the user's and item's next interactions are observed, their embeddings are updated again. We illustrate the model in Figure 2 and the projection operation in Figure 3.

Static and Dynamic Embeddings. Each user and item is assigned two embeddings: a static and a dynamic embedding. We use both embeddings to encode both the long-term stationary properties of the entities and their dynamic properties.

Static embeddings, $\overline{\boldsymbol{u}} \in \mathbb{R}^d \; \forall u \in \mathcal{U}$ and $\overline{\boldsymbol{i}} \in \mathbb{R}^d \; \forall i \in \mathcal{I}$, do not change over time. These are used to express stationary properties such as the long-term interest of users. We use one-hot vectors

[2] Kalman filtering is used to accurately measure the state of a system using a combination of system observations and state estimates given by the laws of the system.

as static embeddings of all users and items, as advised in Time-LSTM [52] and TimeAware-LSTM [7]. Using node2vec [16] gave empirically similar results, so we use one-hot vectors.

On the other hand, each user u and item i is assigned a dynamic embedding represented as $\boldsymbol{u(t)} \in \mathbb{R}^n$ and $\boldsymbol{i(t)} \in \mathbb{R}^n$ at time t, respectively. These embeddings change over time to model their time-varying behavior and properties. The sequence of dynamic embeddings of a user/item is referred to its *trajectory*.

Next, we describe the update and projection operations. Then, we will describe how we predict the future interaction item embeddings and how we train the model.

3.1 Embedding update operation

In the update operation, the interaction $S = (u, i, t, \boldsymbol{f})$ between a user u and item i at time t is used to generate their dynamic embeddings $\boldsymbol{u(t)}$ and $\boldsymbol{i(t)}$. Fig. 2 illustrates the update operations.

Our model uses two recurrent neural networks for updates—RNN_U is shared across all users to update user embeddings, and RNN_I is shared among all items to update item embeddings. The hidden states of the user RNN and the item RNN represent the user and item embeddings, respectively.

The two RNNs are mutually-recursive. When user u interacts with item i, RNN_U updates the embedding $\boldsymbol{u(t)}$ by using the embedding $\boldsymbol{i(t^-)}$ of item i right before time t as an input. $\boldsymbol{i(t^-)}$ is the same as item i's embedding after its previous interaction with any user. Notice that this design decision is in stark contrast with the popular use of items' one-hot vectors to update user embeddings [8, 45, 52], which has the following two disadvantages: (a) one-hot vector only contains the information about the item's id and not the item's current state, and (b) the dimension of the one-hot vector becomes very large when real datasets have millions of items, making the model challenging to train and scale. Instead, we use the dynamic embedding of an item as it reflects the item's current state leading to more meaningful dynamic user embeddings and easier training. For the same reason, RNN_I updates the dynamic embedding $\boldsymbol{i(t)}$ of item i by using the dynamic user embedding $\boldsymbol{u(t^-)}$ (which is u's embedding right before time t). This results in mutually recursive dependency between the embeddings. More formally,

$$\boldsymbol{u(t)} = \sigma(W_1^u \boldsymbol{u(t^-)} + W_2^u \boldsymbol{i(t^-)} + W_3^u \boldsymbol{f} + W_4^u \Delta_u)$$

$$\boldsymbol{i(t)} = \sigma(W_1^i \boldsymbol{i(t^-)} + W_2^i \boldsymbol{u(t^-)} + W_3^i \boldsymbol{f} + W_4^i \Delta_i)$$

where Δ_u denotes the time since u's previous interaction (with any item) and Δ_i is the time since item i's previous interaction (with any user). $\boldsymbol{f}$ is the interaction feature vector. The matrices $W_1^u, \ldots W_4^u$ are the parameters of RNN_U and matrices $W_1^i, \ldots W_4^i$ are the parameters of RNN_I. σ is a sigmoid function to introduce non-linearity. The matrices are trained to predict the embedding of the item at u's next interaction as explained later in Section 3.3.

Variants of RNNs, such as LSTM, GRU, and T-LSTM [52], gave experimentally similar and sometimes worse performance, so we use RNNs in our model to reduce the number of trainable parameters.

3.2 Embedding projection operation

Here we explain one of the major contributions of our algorithm, the embedding projection operator, which predicts the future embedding trajectory of the user. This is done by projecting the embedding of the user at a future time. The projected embedding can then be used for downstream tasks, such as predicting items the user will interact with at a given query/prediction time in the future.

Table 2: Table of symbols used in this paper.

Symbol	Meaning
$\boldsymbol{u(t)}$ and $\boldsymbol{i(t)}$	Dynamic embedding of user u and item i at time t
$\boldsymbol{u(t^-)}$ and $\boldsymbol{i(t^-)}$	Dynamic embedding of user u and item i before time t
$\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{i}}$	Static embedding of user u and item i
$\widehat{\boldsymbol{u}}(\boldsymbol{t})$	Projected embedding of user u at time t
$\widetilde{\boldsymbol{j}}(\boldsymbol{t})$	Predicted item j embedding

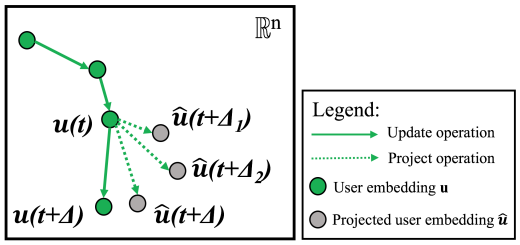

Figure 3: This figure shows the key idea behind *projection operation*. The predicted embedding of user u is shown for different elapsed time $\Delta_1 < \Delta_2 < \Delta$. The predicted embedding drifts farther as more time elapses. When the next interaction is observed, the embedding is updated again.

Figure 3 visualizes the main idea of projecting a user's embedding trajectory. The operation projects the embedding of a user after some time has elapsed since its last interaction at time t. To give an example, a short duration Δ_1 after time t, the user u's projected embedding $\widehat{\boldsymbol{u}}(\boldsymbol{t + \Delta_1})$ is close to its previously observed embedding $\boldsymbol{u(t)}$. As more time $\Delta > \Delta_2 > \Delta_1$ elapses, the projected embeddings drift farther to $\widehat{\boldsymbol{u}}(\boldsymbol{t + \Delta_2})$ and $\widehat{\boldsymbol{u}}(\boldsymbol{t + \Delta})$. When the next interaction is observed at time $t + \Delta$, the user's embedding is updated to $\boldsymbol{u(t + \Delta)}$ using the update operation.

Two inputs are required for the projection operation: u's embedding at time t and the elapsed time Δ. We follow the method suggested in LatentCross [8] to incorporate time into the projected embedding via Hadamard product. We do not simply concatenate the embedding and the time and pass them through a linear layer as prior research has shown that neural networks are inefficient in modeling the interactions between concatenated inputs. Instead, we create a temporal attention vector as described below.

We first convert Δ to a time-context vector $\boldsymbol{w} \in \mathbb{R}^n$ using a linear layer (represented by vector W_p): $\boldsymbol{w} = W_p \Delta$. We initialize W_p by a 0-mean Gaussian. The projected embedding is then obtained as an element-wise product of the time-context vector with the previous embedding as follows:

$$\widehat{\boldsymbol{u}}(\boldsymbol{t + \Delta}) = (1 + \boldsymbol{w}) * \boldsymbol{u(t)}$$

The vector $1 + \boldsymbol{w}$ acts as a temporal attention vector to scale the past user embedding. When $\Delta = 0$, then $\boldsymbol{w} = 0$ and the projected embedding is the same as the input embedding vector. The larger the value of Δ, the more the projected embedding vector differs from the input embedding vector and the projected embedding vector drifts over time.

We find that a linear layer works the best to project the embedding as it is equivalent to a linear transformation in the embedding

space. Adding non-linearity to the transformation makes the projection operation non-linear, which we find experimentally to reduce the prediction performance. Thus, we use the linear transformation as described above.

Next, we describe how we train the model to efficiently project user embeddings such that they are useful in predicting the next item with which the user will interact.

3.3 Training to predict next item embedding

Let u interact with item i at time t and then with item j at time $t + \Delta$. Right before $t + \Delta$, can we predict which item u will interact with? We use this task to train the update and projection operations in *JODIE*. We train *JODIE* to make this prediction using u's projected embedding $\widehat{\boldsymbol{u}}(\boldsymbol{t + \Delta})$.

A crucial design decision here is that *JODIE* directly outputs an item embedding vector, $\tilde{\boldsymbol{j}}(\boldsymbol{t + \Delta})$, instead of an interaction probability between u and item j. This has the advantage of reducing the computation at inference time from linear (in the number of items) to near-constant. Most existing methods [8, 11, 12, 45] that output an interaction probability need to do the expensive neural-network forward pass $|\mathcal{I}|$ times (once for each of item $\in \mathcal{I}$) to find the item with the highest probability score. In contrast, *JODIE* only needs to do forward-pass of the prediction layer once and output a predicted item embedding. Then the item with the closest embedding can be returned in near-constant time by using Locality Sensitive Hashing (LSH) techniques [27]. To maintain the LSH data structure, we update it whenever an item's embedding is updated.

Thus, we train *JODIE* to minimize the L_2 difference between the predicted item embedding $\tilde{\boldsymbol{j}}(\boldsymbol{t + \Delta})$ and the real item embedding $[\bar{\boldsymbol{j}}, \boldsymbol{j}(\boldsymbol{t + \Delta^-})]$ as follows: $||\tilde{\boldsymbol{j}}(\boldsymbol{t + \Delta}) - [\bar{\boldsymbol{j}}, \boldsymbol{j}(\boldsymbol{t + \Delta^-})]||_2$. Here, $[\boldsymbol{x}, \boldsymbol{y}]$ represents the concatenation of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and the superscript '-' indicates the embedding immediately before the time.

We make this prediction using the projected user embedding $\widehat{\boldsymbol{u}}(\boldsymbol{t + \Delta})$ and the embedding $\boldsymbol{i}(\boldsymbol{t + \Delta^-})$ of item i (the item from u's previous interaction) immediately before time $t + \Delta$. The reason we include $\boldsymbol{i}(\boldsymbol{t + \Delta^-})$ is two-fold: (a) i may interact with other users between time t and $t + \Delta$, and thus the embedding contains more recent information, and (b) users often interact with the same item consecutively (i.e., $i = j$) and including the item embedding helps to ease the prediction. We use both the static and dynamic embeddings to predict the static and dynamic embedding of the predicted item j. The prediction is made using a fully connected linear layer as follows:

$$\tilde{\boldsymbol{j}}(\boldsymbol{t + \Delta}) = W_1 \widehat{\boldsymbol{u}}(\boldsymbol{t + \Delta}) + W_2 \bar{\boldsymbol{u}} + W_3 \boldsymbol{i}(\boldsymbol{t + \Delta^-}) + W_4 \bar{\boldsymbol{i}} + B$$

where $W_1, \ldots W_4$ and the bias vector B make the linear layer.

Training the model. *JODIE* is trained to minimize the L_2 distance between the predicted item embedding and the ground truth item's embedding at every interaction. We calculate the total loss as follows:

$$\begin{aligned} Loss = & \sum_{(u,j,t,f) \in S} ||\tilde{\boldsymbol{j}}(\boldsymbol{t}) - [\bar{\boldsymbol{j}}, \boldsymbol{j}(\boldsymbol{t^-})]||_2 \\ & + \lambda_U ||\boldsymbol{u}(\boldsymbol{t}) - \boldsymbol{u}(\boldsymbol{t^-})||_2 \ + \ \lambda_I ||\boldsymbol{j}(\boldsymbol{t}) - \boldsymbol{j}(\boldsymbol{t^-})||_2 \end{aligned} \tag{1}$$

The first loss term minimizes the predicted embedding error. The last two terms are added to regularize the loss and prevent the consecutive dynamic embeddings of a user and item to vary too much, respectively. λ_U and λ_I are scaling parameters to ensure the losses are in the same range. It is noteworthy that we do not use negative sampling during training as *JODIE* directly outputs the embedding of the predicted item.

Extending the loss for categorical prediction. In certain prediction tasks, such as user state change prediction, additional training labels may be present for supervision. The user state change labels are binary (categorical). In those cases, we can train another prediction function $\Theta : \mathbb{R}^{n+d} \rightarrow C$ to predict the label using the embedding of the user after an interaction. We calculate the cross-entropy loss for categorical labels and add the loss to the above loss function with another scaling parameter. We explicitly do not just train to minimize only the cross-entropy loss to prevent overfitting.

3.4 t-Batch: Training data batching

Here we explain the batching algorithm we propose to parallelize the training of *JODIE*. It is important to maintain temporal dependencies between interactions during training, such that interaction S_r is processed before S_k $\forall r < k$.

Existing methods that use a single RNN, such as T-LSTM [52] and RRN [8], split users into different batches and process them in parallel. This is possible because these approaches use one-hot vector encodings of items as inputs and can thus be trained using the standard Back Propagation Through Time (BPTT) mechanism.

However, in *JODIE*, the mutually-recursive RNNs enable us to incorporate the item's embedding to update the user embedding and vice-versa. This creates interdependencies between two users that interacted with the same item and this prevents us from simply splitting users into separate batches and processing them in parallel.

Most existing methods that also use two mutually-recursive RNNs [11, 49] naively process all the interactions one at a time in sequential order. However, this is not scalable to a large number of interactions as the training process is very slow. Therefore, we train *JODIE* using a training data batching algorithm that we call *t-Batch*. This leads to an order of magnitude of speed-up in *JODIE* compared to most existing training approaches.

Creating the training batches is challenging because it has two requirements: (1) all interactions in each batch should be processed in parallel, and (2) processing the batches in increasing order of their index should maintain the temporal ordering of the interactions and thus, it should generate the same embedding as no batching.

To overcome these challenges, *t-Batch* creates each batch by selecting independent edge sets of the interaction network, i.e., two interactions in the same batch do not share any common user or item. *JODIE* works iteratively in two steps: the select step and the reduce step. In the *select step*, a new batch is created by selecting the maximal edge set such that each edge (u, i) is the lowest time-stamped edge incident on both u and i. This automatically makes the batch an independent edge set. In the *reduce step*, the selected edges are removed from the network. *JODIE* iterates the two steps till no edges remain in the graph. This way the trajectory of each user is split

In practice, we implement *t-Batch* as a sequential algorithm as follows. We assign each interaction S_r to a batch B_k, where $k \in [1, |\mathcal{I}|]$. We initialize $|\mathcal{I}|$ empty batches (in the worst case scenario that each batch only has one interaction). We iterate through the temporally-sorted sequence of interactions $S_1 \ldots S_{|\mathcal{I}|}$ and add each interaction

to a batch B_k. Let $maxBatch(e, r)$ be the batch with the largest index that has an interaction involving an entity e till interaction S_r. Then, the interaction S_{r+1} (say, between user u and item i) is assigned to the batch with index = $max(1 + maxBatch(u, r), 1 + maxBatch(i, r))$. The complexity of creating the batches is $O(|S|)$, i.e., linear in the number of interactions, as each interaction is looked at once.

It is trivial to verify that *t-Batch* satisfies the two requirements. *t-Batch* ensures that each user and item appears at most once in every batch and thus, each batch can be parallelized. In addition, the r^{th} and $r + 1^{st}$ interactions of every user and every item are assigned to batches B_k and B_l, respectively, such that $k < l$. So, *JODIE* can process the batches in increasing order of their indices to ensure that the temporal ordering of the transactions is respected and all trajectories are maintained.

We do not predetermine the number and size of the batches because it depends on the interactions in the dataset. The number of batches can range between 1 and $|\mathcal{I}|$. Let us illustrate these two extreme cases. When all interactions have unique users and items, then only one batch is created that has all the interactions. On the other extreme, if all interactions are associated to the same user or the same item, then $|\mathcal{I}|$ batches are created. Therefore, we initialize $|\mathcal{I}|$ batches and discard all trailing empty batches after assignment.

3.5 Differences between *JODIE* and DeepCoevolve

DeepCoevolve is the closest state-of-the-art algorithm to *JODIE* because it also trains two mutually-recursive RNNs to generate embedding trajectories. However, the key differences between *JODIE* and DeepCoevolve are the following: (i) *JODIE* uses a novel project function to predict the future trajectory of users. Instead, DeepCoevolve maintains the same embedding of a user between two of its consecutive interactions. Predicting the trajectory enables *JODIE* to make more effective predictions. (ii) *JODIE* predicts the embedding of the next item that a user will interact with. In contrast, DeepCoevolve predicts the probability of interaction between a user and an item. During inference time, DeepCoevolve requires $|\mathcal{I}|$ forward passes through the inference layer (for $|\mathcal{I}|$ items) to recommend the item with the highest score. On the other hand, *JODIE* takes near-constant time. (iii) *JODIE* is trained with batches of interaction data, as opposed to individual interactions.

As a result, as we will see in the experiments section, *JODIE* significantly outperforms DeepCoevolve both in terms of performance and training time. *JODIE* is 9.2× faster, 45% better in predicting future interactions, and 13.9% better in predicting user state change on average.

4 EXPERIMENTS

In this section, we experimentally validate the effectiveness of *JODIE* on two tasks: future interaction prediction and user state change prediction. We conduct experiments on three datasets each and compare with six strong baselines to show the following:

(1) *JODIE* outperforms the baselines by at least 20% in terms of mean reciprocal rank in predicting the next item and 12% on average in predicting user state change.
(2) We show that *JODIE* is 9.2× faster than DeepCoevolve and comparable to other baselines.
(3) *JODIE* is robust in performance to the availability of training data and the dimension of the embedding.
(4) Finally, in a case study on the MOOC dataset, we show that *JODIE* can predict student drop-out five interactions in advance.

We first explain the experimental setting and the baseline methods and then describe the experimental results.

Experimental setting. We train all models by splitting the data by time to simulate the real situation. Thus, we train all models on the first τ% interactions, validate on the next τ_v%, and test on the last remaining interactions.

For a fair comparison, we use 128 dimensions as the dimensionality of the dynamic embedding for all algorithms and one-hot vectors for static embeddings. All algorithms are run for 50 epochs, and all reported numbers for all models are for the test data corresponding to the best performing validation set.

Baselines. We compare *JODIE* with six state-of-the-art algorithms spanning three algorithmic categories:

(1) **Deep recurrent recommender models:** in this category, we compare with RRN [45], LatentCross [8], Time-LSTM [52], and standard LSTM. These algorithms are state-of-the-art in recommender systems and generate dynamic user embeddings. We use Time-LSTM-3 cell for Time-LSTM as it performs the best in the original paper [52], and LSTM cells in RRN and LatentCross models. As is standard, we use the one-hot vector of items as inputs to these models.
(2) **Dynamic co-evolution models:** here we compare with the state-of-the-art algorithm, DeepCoevolve [11], which has been shown to outperform other co-evolutionary point-process algorithms [41, 44]. We use 10 negative samples per interaction for computational tractability.
(3) **Temporal network embedding models:** we compare *JODIE* with CTDNE [33] which is the state-of-the-art in generating embeddings from temporal networks. As it generates static embeddings, we generate new embeddings after each edge is added. We use uniform sampling of neighborhood as it performs the best in the original paper [33].

4.1 Experiment 1: Future interaction prediction

The prediction task here is: given all interactions till time t, which item will user u interact with at time t (out of all $|\mathcal{I}|$ items)?

We use three datasets in this experiments:

• **Reddit post dataset:** this public dataset consists of one month of posts made by users on subreddits [2]. We selected the 1,000 most active subreddits as items and the 10,000 most active users. This results in 672,447 interactions. We convert the text of each post into a feature vector representing their LIWC categories [35].

• **Wikipedia edits:** this public dataset is one month of edits made by edits on Wikipedia pages [3]. We selected the 1,000 most edited pages as items and editors who made at least 5 edits as users (a total of 8,227 users). This generates 157,474 interactions. Similar to the Reddit dataset, we convert the edit text into a LIWC-feature vector.

• **LastFM song listens:** this public dataset has one month of who-listens-to-which song information [18]. We selected all 1000 users and the 1000 most listened songs resulting in 1,293,103 interactions. In this dataset, interactions do not have features.

Table 3: Future interaction prediction experiment: Table comparing the performance of *JODIE* with state-of-the-art algorithms, in terms of mean reciprocal rank (MRR) and recall@10. The best algorithm in each column is colored blue and second best is light blue. The last two columns show the minimum percentage improvement of *JODIE* over the method, across over all datasets. We see that *JODIE* outperforms all baselines by at least 20% in MRR and 14% in recall@10.

Method	Reddit		Wikipedia		LastFM		Minimum % improvement of *JODIE* over method	
	MRR	Recall@10	MRR	Recall@10	MRR	Recall@10	MRR	Recall@10
LSTM [52]	0.355	0.551	0.329	0.455	0.062	0.119	104.5%	54.6%
Time-LSTM [52]	0.387	0.573	0.247	0.342	0.068	0.137	87.6%	48.7%
RRN [45]	0.603	0.747	0.522	0.617	0.089	0.182	20.4%	14.1%
LatentCross [8]	0.421	0.588	0.424	0.481	0.148	0.227	31.8%	35.2%
CTDNE [33]	0.165	0.257	0.035	0.056	0.01	0.01	340.0%	231.5%
DeepCoevolve [11]	0.171	0.275	0.515	0.563	0.019	0.039	44.8%	46.0%
JODIE (proposed)	**0.726**	**0.852**	**0.746**	**0.822**	**0.195**	**0.307**	-	-

Table 4: User state change prediction: Table comparing the performance in terms of AUC of *JODIE* with state of the art algorithms. The best algorithm in each column is colored blue and the second best is light blue. *JODIE* outperforms the baselines by at least 12.63% on average.

Method	Reddit	Wikipedia	MOOC	Mean improvement of *JODIE*
LSTM	0.523	0.575	0.686	23.08%
Time-LSTM	0.556	0.671	0.711	12.63%
RRN	0.586	0.804	0.558	13.69%
LatentCross	0.574	0.628	0.686	15.62%
DeepCoevolve	0.577	0.663	0.671	13.94%
JODIE	**0.599**	**0.831**	**0.756**	-

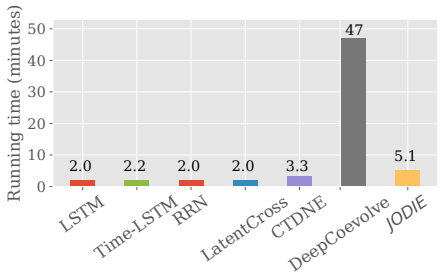

Figure 4: Figure compares the running time of *JODIE* and all baselines on the Reddit dataset. *JODIE* is 9.2× faster than DeepCoevolve and is comparable to the other baselines.

We select these datasets such that they vary in terms of users' repetitive behavior: in Wikipedia and Reddit, a user interacts with the same item consecutively in 79% and 61% interactions, respectively, while in LastFM, this happens in only 8.6% interactions.

Experimental setting. We use the first 80% data to train, next 10% to validate, and the final 10% to test. We measure the performance of the algorithms in terms of the mean reciprocal rank (MRR) and recall@10—MRR is the average of the reciprocal rank and recall@10 is the fraction of interactions in which the ground truth item is ranked in the top 10. Higher values for both are better. For every interaction, the ranking of ground truth item is calculated with respect to all the items in the dataset.

For *JODIE*, items are ranked based on their L_2 distance from the predicted item embedding. The rank of the ground truth item is calculated in this ranked list.

Results. Table 3 compares the results of *JODIE* with the six state-of-the-art methods. We observe that *JODIE* significantly outperforms all baselines in all datasets across both metrics on the three datasets. Among the baselines, there is no clear winner—while RRN performs the better in Reddit and Wikipedia, LatentCross performs better in LastFM. As CTDNE generates static embedding, its performance is low. We calculate the percentage improvement of *JODIE* over the baseline as (performance of *JODIE* minus performance of baseline)/(performance of baseline). Across all datasets, the minimum improvement of *JODIE* is at least 20% in terms of MRR and 14% in terms of recall@10. Please note that *JODIE* outperforms DeepCoevolve, the closest baseline in terms of the algorithm, by at least 44.8% in MRR across all datasets.

Noticeably, we observe that *JODIE* performs well irrespective of how repetitive users are—the MRR at least 20.4% higher in Wikipedia and Reddit (high repetition datasets), and at least 31.75% higher in LastFM (low repetition dataset). This means *JODIE* is able to learn to balance personal preference with users' non-repetitive interaction behavior.

4.2 Experiment 2: User state change prediction

In this experiment, the task is to predict if an interaction will lead to a state change in user, particularly in two use cases: predicting if a user will be banned and predicting if a student will drop-out of a course. Till a user is banned or drops-out, the label of the user is '0', and their last interaction has the label '1'. For users that are not banned or do not drop-out, the label is always '0'. This is a highly challenging task as less than 1% of the labels are '1'.

We use three datasets for this task:

• **Reddit bans:** Reddit post dataset (from Section 4.1) with ground-truth labels of banned users from Reddit This gives 366 true labels among 672,447 interactions (= 0.05%).

• **Wikipedia bans:** Wikipedia edit data (from Section 4.1) with public ground-truth labels of banned users [3]. This results in 217 positive labels among 157,474 interactions (= 0.14%).

• **MOOC student drop-out:** this public dataset consists of actions, e.g., viewing a video, submitting an answer, etc., done by students on a MOOC online course [1]. This dataset consists of 7,047 users interacting with 98 items (videos, answers, etc.) resulting in over 411,749 interactions. There are 4,066 drop-out events (= 0.98%).

Experimental setting. In this experiment, we train the models on the first 60% interactions, validate on the next 20%, and test on the last 20% interactions. We evaluate the models using the area

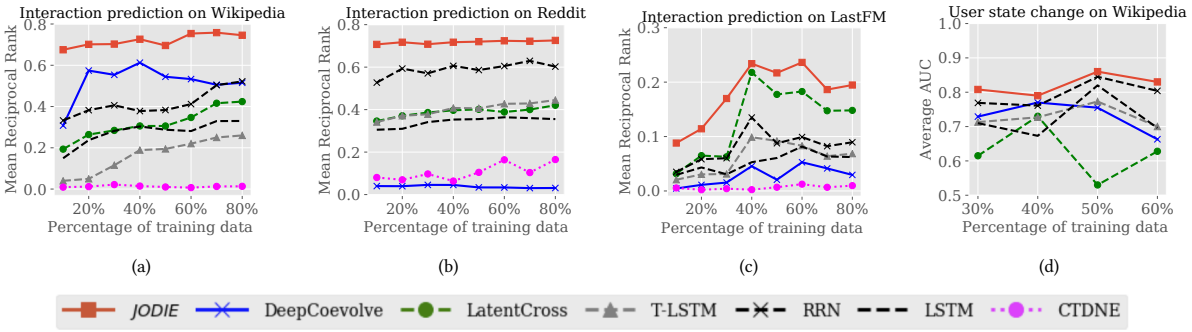

Figure 5: Robustness of *JODIE*: Figures (a–c) compare the mean reciprocal rank (MRR) of *JODIE* with baselines on interaction prediction task, by varying the training data size. Figure (d) shows the AUC of user state change prediction task by varying the training data size. We see *JODIE* consistently has the highest scores.

under the curve metric (AUC), a standard metric in the tasks with highly imbalanced labels.

For the baselines, we train a logistic regression classifier on the training data using the dynamic user embedding as input. As always, for all models, we report the test AUC for the epoch with the highest validation AUC.

Results. Table 4 compares the performance of *JODIE* on the three datasets with the baseline models. We see that *JODIE* outperforms the baselines by at least 12% on average in predicting user state change across all datasets. *JODIE* outperforms RRN, the closest competitor in the ban prediction task, by at least 2.2% while it outperforms RRN by 28% in the student drop-out task. Note that DeepCoevolve, which is the most similar baseline algorithmically, is outperformed by 13.9% by *JODIE* on average. Thus, *JODIE* consistently performs the best across various datasets.

4.3 Experiment 3: Runtime experiment

Here we compare the running time of *JODIE* with the baseline algorithms. Algorithmically, the DeepCoevolve is the closest to *JODIE* as it also trains two mutually-recursive RNNs. The other methods train only one RNN and are therefore easily scalable.

Figure 4 shows the running time (in minutes) of one epoch of the Reddit dataset.[3] We find that *JODIE* is 9.2× faster than DeepCoevolve (its closest algorithmic competitor). At the same time, the running time of *JODIE* is comparable to the other baselines that only use one RNN in their model. This shows that *JODIE* is able to train the mutually-recursive model in equivalent time as non-mutually-recursive models, because of the use of the *t-Batch* training batching algorithm.

In addition, we find that *JODIE* without *t-Batch* took 43.53 minutes while *JODIE* with *t-Batch* took 5.13 minutes. Thus, *t-Batch* results in 8.4× speed-up.

4.4 Experiment 4: Robustness to the proportion of training data

In this experiment, we validate the robustness of *JODIE* by varying the percentage of training data and comparing the performance of the algorithms in both the tasks of future interaction prediction and user state change prediction.

[3]We ran the experiment on one NVIDIA Titan X Pascal GPUs with 12Gb of RAM at 10Gbps speed.

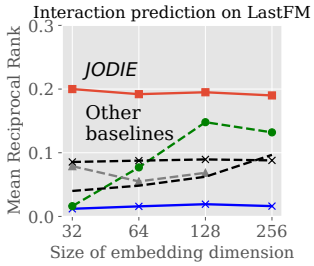

Figure 6: Robustness to dynamic embedding size: The performance of *JODIE* is stable with the change in dynamic embedding size, for the task of interaction prediction on LastFM dataset. Please refer to the legend in Figure 5.

For the next item prediction, we vary the training data percentage from 10% to 80%. In each case, we take the 10% interactions after the training data as validation and the next 10% interactions next as testing. This is done to compare the performance on the same testing data size. Figures 5(a–c) show the change in mean reciprocal rank (MRR) of all the algorithms on the three datasets, as the training data size is increased. We note that the performance of *JODIE* is stable and does not vary much across the data points. Moreover, *JODIE* consistently outperforms the baseline models by a significant margin (by a maximum of 33.1%).

We make similar observations in user state change prediction task. Here, we vary training data percents to 20%, 40%, and 60%, and in each case take the following 20% interactions as validation and the next 20% interactions as the test. Figure 5(d) shows the AUC of all the algorithms on the Wikipedia dataset. Other datasets have similar results. Again, we find that *JODIE* is stable and consistently outperforms the baselines, irrespective of the training data size.

4.5 Experiment 5: Embedding size

Finally, we validate the effect of the dynamic embedding size on the predictions. To do this, we vary the dynamic embedding dimension from 32 to 256 and calculate the mean reciprocal rank for interaction prediction on the LastFM dataset. The effect on other datasets is similar. The resulting figure is showing in Figure 6. We find that the embedding dimension size has little effect on the performance of *JODIE* and it performs the best overall. Interestingly, improvement in *JODIE* is higher for smaller embedding dimensions. This is

because JODIE uses both the static and the dynamic embedding for prediction, which gives it the power to learn from both parts.

5 CONCLUSIONS

In this paper, we proposed a coupled recurrent neural network model called JODIE that learns dynamic embeddings of users and items from a sequence of temporal interactions. JODIE learns to predict the future embeddings of users and items, which leads it to give better prediction performance of future user-item interactions and change in user state. We also presented a training data batching method that makes JODIE an order of magnitude faster than similar baselines.

There are several directions for future work. Learning embeddings for individual users and items is expensive, and one could learn trajectories for groups of users or items to reduce the number of parameters. Another direction is characterizing the trajectories to cluster similar entities. Finally, an innovative direction would be to design new items based on missing predicted items that many users are likely to interact with.

Acknowledgements. JL is a Chan Zuckerberg Biohub investigator. This research has been supported in part by NSF OAC-1835598, DARPA MCS, DARPA ASED, ARO MURI, Amazon, Boeing, Docomo, Hitachi, JD, Siemens, and Stanford Data Science Initiative.

REFERENCES

[1] Kdd cup 2015. https://biendata.com/competition/kddcup2015/data/.
[2] Reddit data dump. http://files.pushshift.io/reddit/.
[3] Wikipedia edit history dump. https://meta.wikimedia.org/wiki/Data_dumps.
[4] D. Agrawal, C. Budak, A. El Abbadi, T. Georgiou, and X. Yan. Big data in online social networks: user interaction analysis to model user behavior in social networks. In *DNIS*, 2014.
[5] T. Arnoux, L. Tabourier, and M. Latapy. Combining structural and dynamic information to predict activity in link streams. In *ASONAM*, 2017.
[6] T. Arnoux, L. Tabourier, and M. Latapy. Predicting interactions between individuals with structural and dynamical information. *CoRR*, 2018.
[7] I. M. Baytas, C. Xiao, X. Zhang, F. Wang, A. K. Jain, and J. Zhou. Patient subtyping via time-aware lstm networks. In *KDD*, 2017.
[8] A. Beutel, P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, and E. H. Chi. Latent cross: Making use of context in recurrent recommender systems. In *WSDM*, 2018.
[9] J. Cheng, M. Bernstein, C. Danescu-Niculescu-Mizil, and J. Leskovec. Anyone can become a troll: Causes of trolling behavior in online discussions. In *CSCW*, 2017.
[10] J. Cheng, C. Lo, and J. Leskovec. Predicting intent using activity logs: How goal specificity and temporal range affect user behavior. In *WWW*, 2017.
[11] H. Dai, Y. Wang, R. Trivedi, and L. Song. Deep coevolutionary network: Embedding user and item features for recommendation. *arXiv:1609.03675*, 2016.
[12] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song. Recurrent marked temporal point processes: Embedding event history to vector. In *KDD*, 2016.
[13] M. Farajtabar, Y. Wang, M. Gomez-Rodriguez, S. Li, H. Zha, and L. Song. COEVOLVE: A joint point process model for information diffusion and network co-evolution. In *NeurIPS*, 2015.
[14] P. Goyal and E. Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge Based Systems*, 151:78–94, 2018.
[15] P. Goyal, N. Kamra, X. He, and Y. Liu. Dyngem: Deep embedding method for dynamic graphs. *arXiv:1805.11273*, 2018.
[16] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.
[17] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 40(3):52–74, 2017.
[18] B. Hidasi and D. Tikk. Fast als-based tensor factorization for context-aware recommendation from implicit feedback. In *ECML*, 2012.
[19] T. Iba, K. Nemoto, B. Peters, and P. A. Gloor. Analyzing the creative editing behavior of wikipedia editors: Through dynamic social network analysis. *Procedia-Social and Behavioral Sciences*, 2(4):6441–6456, 2010.
[20] S. J. Julier and J. K. Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–194, 1997.
[21] R. R. Junuthula, M. Haghdan, K. S. Xu, and V. K. Devabhaktuni. The block point process model for continuous-time event-based dynamic networks. *CoRR*, 2017.
[22] R. R. Junuthula, K. S. Xu, and V. K. Devabhaktuni. Leveraging friendship networks for dynamic link prediction in social interaction networks. In *ICWSM*, 2018.
[23] M. Kloft, F. Stiehler, Z. Zheng, and N. Pinkwart. Predicting mooc dropout over weeks using machine learning methods. In *EMNLP*, 2014.
[24] S. Kumar, W. L. Hamilton, J. Leskovec, and D. Jurafsky. Community interaction and conflict on the web. In *The World Wide Web Conference*, 2018.
[25] S. Kumar, B. Hooi, D. Makhija, M. Kumar, C. Faloutsos, and V. Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In *WSDM*, 2018.
[26] S. Kumar, F. Spezzano, and V. Subrahmanian. Vews: A wikipedia vandal early warning system. In *KDD*, 2015.
[27] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*. Cambridge university press, 2014.
[28] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu. Attributed network embedding for learning in a dynamic environment. In *CIKM*, 2017.
[29] T. Li, J. Zhang, P. S. Yu, Y. Zhang, and Y. Yan. Deep dynamic network embedding for link prediction. *IEEE Access*, 6:29219–29230, 2018.
[30] X. Li, N. Du, H. Li, K. Li, J. Gao, and A. Zhang. A deep learning approach to link prediction in dynamic networks. In *SDM*, 2014.
[31] T. R. Liyanagunawardena, A. A. Adams, and S. A. Williams. Moocs: A systematic study of the published literature 2008-2012. *The International Review of Research in Open and Distributed Learning*, 14(3):202–227, 2013.
[32] Y. Ma, Z. Guo, Z. Ren, Y. E. Zhao, J. Tang, and D. Yin. Dynamic graph neural networks. *CoRR*, abs/1810.10627, 2018.
[33] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim. Continuous-time dynamic network embeddings. In *WWW BigNet workshop*, 2018.
[34] R. Pálovics, A. A. Benczúr, L. Kocsis, T. Kiss, and E. Frigó. Exploiting temporal influence in online recommendation. In *RecSys*, 2014.
[35] J. W. Pennebaker, M. E. Francis, and R. J. Booth. Linguistic inquiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, 71(2001):2001, 2001.
[36] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*, 2018.
[37] V. Raghavan, G. Ver Steeg, A. Galstyan, and A. G. Tartakovsky. Modeling temporal activity patterns in dynamic social networks. *IEEE TCSS*, 1(1):89–107, 2014.
[38] M. Rahman, T. K. Saha, M. A. Hasan, K. S. Xu, and C. K. Reddy. Dylink2vec: Effective feature representation for link prediction in dynamic networks. *CoRR*, 2018.
[39] S. Sajadmanesh, J. Zhang, and H. R. Rabiee. Continuous-time relationship prediction in dynamic heterogeneous information networks. *CoRR*, 2017.
[40] S. Sedhain, S. Sanner, L. Xie, R. Kidd, K. Tran, and P. Christen. Social affinity filtering: recommendation through fine-grained analysis of user interactions and activities. In *COSN*, 2013.
[41] R. Trivedi, H. Dai, Y. Wang, and L. Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *ICML*, 2017.
[42] R. Trivedi, M. Farajtbar, P. Biswal, and H. Zha. Representation learning over dynamic graphs. *arXiv:1803.04051*, 2018.
[43] P. B. Walker, S. G. Fooshee, and I. Davidson. Complex interactions in social and event network analysis. In *SBP-BRiMS*, 2015.
[44] Y. Wang, N. Du, R. Trivedi, and L. Song. Coevolutionary latent feature processes for continuous-time user-item interactions. In *NeurIPS*, 2016.
[45] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing. Recurrent recommender networks. In *WSDM*, 2017.
[46] D. Yang, T. Sinha, D. Adamson, and C. P. Rosé. Turn on, tune in, drop out: Anticipating student dropouts in massive open online courses. In *NeurIPS Data-driven education workshop*, 2013.
[47] J. You, Y. Wang, A. Pal, P. Eksombatchai, C. Rosenburg, and J. Leskovec. Hierarchical temporal convolutional networks for dynamic recommender systems. In *The World Wide Web Conference*, 2019.
[48] S. Zhang, L. Yao, and A. Sun. Deep learning based recommender system: A survey and new perspectives. *arXiv:1707.07435*, 2017.
[49] Y. Zhang, Y. Xiong, X. Kong, and Y. Zhu. Learning node embeddings in interaction graphs. In *CIKM*, 2017.
[50] L.-k. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang. Dynamic network embedding by modeling triadic closure process. In *AAAI*, 2018.
[51] L. Zhu, D. Guo, J. Yin, G. Ver Steeg, and A. Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE TKDE*, 28(10):2765–2777, 2016.
[52] Y. Zhu, H. Li, Y. Liao, B. Wang, Z. Guan, H. Liu, and D. Cai. What to do next: modeling user behaviors by time-lstm. In *IJCAI*, 2017.

A APPENDIX

Here we describe the technical details of the model as required as part of the submission.

The code and datasets are available on the project website: `https://snap.stanford.edu/jodie`.

Table 5: Table with dataset information.

Data	Users	Items	Interactions	Bans	Action Repetition
Reddit	10,000	1,000	672,447	366	79%
Wikipedia	8,227	1,000	157,474	217	61%
LastFM	1,000	1,000	1,293,103	-	8.6%
MOOC	7,047	98	411,749	4,066	-

Table 6: Table with model parameters.

Parameter	Value
Optimizer	Adam
Learning rate	1e-3
Model weight decay	1e-5
Dynamic embedding size	128
Number of epochs	50
Future interaction prediction experiment	
Training data percent	80%
Validation data percent	10%
Test data percent	10%
User state change experiment	
Training data percent	60%
Validation data percent	20%
Test data percent	20%

We coded all the models and the baselines in PyTorch. Table 5 mentions the dataset details and Table 6 mentions the model parameters.